



**Escola Politècnica Superior  
d'Enginyeria de Vilanova i la Geltrú**

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TREBALL FINAL DE GRAU

**TÍTOL: CREACIÓ DE UN CLÚSTER/CLOUD A PARTIR DE RASPBERRY  
PI 2 PARA UN SISTEMA DISTRIBUIDO**

**AUTORS: HERNÁNDEZ VILLAHERMOSA, XAVIER**

**DATA DE PRESENTACIÓ: OCTUBRE, 2016**

<b>COGNOMS:</b>	<b>NOM:</b>
<b>TITULACIÓ:</b>	
<b>PLA:</b>	
<b>DIRECTOR:</b>	
<b>DEPARTAMENT:</b>	

**QUALIFICACIÓ DEL TFG**

<b><u>TRIBUNAL</u></b>		
<b>PRESIDENT</b>	<b>SECRETARI</b>	<b>VOCAL</b>
<b>DATA DE LECTURA:</b>		

**Aquest Projecte té en compte aspectes mediambientals:**  Sí  No

## RESUMEN

Este trabajo se ha llevado a cabo dentro del marco tecnológico de las *Smart City*. Los objetivos principales del proyecto son plantear una estructura teórica de sistema distribuido mediante placas *Raspberry Pi 2* para una *Smart City*, y la realización experimental de pruebas sobre las placas mencionadas para soportar distintas funcionalidades necesarias para un sistema distribuido. En la memoria del proyecto se realiza un estudio teórico del *hardware* necesario para implementar un sistema distribuido en una *Smart City*, comparando los dispositivos *hub*, *switch* y *router* como distribuidores de los datos para la comunicación entre las máquinas. Además, se examina que tipo de tipología o esquema es más eficiente para un sistema distribuido, entre *grid computing*, *clúster computing* o *cloud computing*.

Para comprobar la capacidad de las *Raspberry* para soportar las características de un sistema distribuido se han realizado las siguientes pruebas experimentales: emulación de una arquitectura *x86*, concretamente el sistema operativo *Windows 98*; instalación de un *software* potente de *cloud computing*, para ello se ha configurado la tecnología de *OpenStack*, y la implementación de un sistema distribuido en *clúster* entre dos placas, para la posterior construcción de un programa mediante el estándar *Message Passing Protocol* para distribuir la carga de datos.

Se ha podido concluir que el potencial de las máquinas es muy bajo para virtualizar o soportar un *software* con todos los requisitos que comporta el *cloud computing*. Como alternativa, se propone implementar una infraestructura en red compuesta de *routers* para la comunicación entre nodos *Raspberry Pi 2*, y estos nodos conectados a distintos sensores obteniendo datos del entorno. Las placas realizarían los cálculos pertinentes y los resultados se mandarían a un Data Center. Posteriormente desde otros servidores se podrían utilizar esos datos para dar servicios al ciudadano.

### Paraules clau (màxim 10):

Smart City	<i>Raspberry Pi 2</i>	<i>hub</i>	<i>switch</i>
<i>router</i>	<i>grid computing</i>	<i>clúster computing</i>	<i>cloud computing</i>
<i>Message Passing Protocol</i>	<i>OpenStack</i>		

# ÍNDICE

1. Introducción.....	8
1.1. <i>Smart City</i> .....	9
1.2. Objetivos.....	10
2. Hardware.....	11
2.1. Arquitectura <i>RISC</i> / Arquitectura <i>CISC</i> .....	12
2.2. Características <i>Raspberry Pi 2</i> .....	14
2.3. <i>Hub, Switch y Router</i> .....	17
2.3.1. Decisión .....	21
3. Software.....	21
3.1. Emulación y virtualización .....	22
3.1.1. Primera implementación práctica: Emulación x86 sobre <i>ARM</i> .....	23
3.1.2. Resultados.....	25
3.2. Modelos de Sistema Distribuido.....	25
3.2.1. Alternativas de <i>Cloud Computing</i> .....	35
3.2.2. Introducción <i>OpenStack</i> .....	37
3.2.3. Segunda Implementación práctica: <i>OpenStack</i> sobre <i>Raspberry Pi</i> 2.....	44
3.2.4. Resultados.....	50
3.3. Ejecución paralela en clúster.....	56
3.3.1. Introducción <i>MPI</i> .....	58
3.3.2. Tercera Implementación práctica: <i>MPI</i> sobre <i>Raspberry Pi 2</i> .....	64
3.3.3. Resultados.....	67
4. Conclusiones.....	69
5. Agradecimientos.....	70
6. Bibliografía.....	71
7. Anexo.....	73

## SUMARIO DE FIGURAS

Figura 1: Imagen de la placa <i>Single Board Computer Raspberry Pi 2</i>	11
Figura 2: Infografía donde se indican las partes esenciales de la <i>Raspberry Pi 2</i> .	15
Figura 3: Infografía explicativa de la estructura en capas del modelo OSI.	17
Figura 4: Infografía explicativa del funcionamiento de un <i>switch</i> y un <i>router</i> .	18
Figura 5: Infografía informativa de los campos que forman el paquete <i>IP</i> .	19
Figura 6: Infografía explicativa del flujo de envío de paquetes por parte del <i>router</i> .	20
Figura 7: Figura ilustrativa del modelo <i>Infraestructure as a Service</i>	30
Figura 8: Figura ilustrativa del modelo <i>Platform as a Service</i>	30
Figura 9: Figura ilustrativa del modelo <i>Software as a Service</i>	30
Figura 10: Alternativas de <i>Cloud Computing</i> : <i>OpenStack</i> , <i>Eucalyptus</i> , <i>OpenNebula</i> y <i>CloudStack</i> .	36
Figura 11: Esquema conceptual de la distribución de los módulos <i>OpenStack</i> .	43
Figura 12: Captura de pantalla donde se muestra la incompatibilidad con <i>Raspbian</i> .	45
Figura 13: Captura de pantalla donde se muestra errores con las dependencias.	46
Figura 14: Captura de pantalla donde se muestra la incompatibilidad entre <i>Liberty</i> y <i>Ubuntu Trusty</i> .	46
Figura 15: Captura de pantalla donde se muestra el consumo de la memoria <i>RAM</i> por parte de los módulos de <i>OpenStack</i> .	47
Figura 16: Captura de pantalla donde se muestra el consumo de la <i>CPU</i> por parte de los módulos de <i>OpenStack</i> .	48
Figura 17: Captura de pantalla donde se muestra el consumo de la memoria <i>RAM</i> por parte de los módulos de <i>OpenStack</i> después de ampliar la <i>swap</i> .	49
Figura 18: Captura de pantalla donde se muestra el consumo de la memoria <i>RAM</i> por parte de los módulos de <i>OpenStack</i> después de ampliar la <i>swap</i> .	49
Figura 19: Captura de pantalla donde se muestra los servicios del módulo <i>Nova</i> .	51
Figura 20: Captura de pantalla donde se muestra el error al intentar crear una máquina virtual en <i>OpenStack</i> .	51
Figura 21: Login para la <i>API OpenStack</i> .	52
Figura 22: Servicios de gestión de la <i>API OpenStack</i> .	52
Figura 23: Servicio <i>nova-compute</i> del módulo <i>Nova</i> encendido.	53
Figura 24: Servicio <i>nova-compute</i> del módulo <i>Nova</i> apagado.	53
Figura 25: Servicio gestor de imágenes <i>Glance</i> .	54
Figura 26: Infraestructura virtual para el aprovisionamiento de máquinas.	54
Figura 27: Captura de pantalla donde se muestra los nodos con hipervisores.	55
Figura 28: Captura de pantalla donde se muestra el error al crear la máquina virtual desde la <i>API</i> .	55
Figura 29: Captura de pantalla donde se muestra la división de un bloque en procesos.	57
Figura 30: Figura que muestra una multiplicación de dos matrices cuadradas de 3x3.	65

## SUMARIO DE TABLAS

Tabla 1: Tabla comparativa <i>RISC</i> vs <i>CISC</i> .	13
Tabla 2: Tabla con especificaciones técnicas de La <i>Raspberry Pi 2 model B</i> .	17
Tabla 3: Tabla comparativa de <i>Clúster Computing</i> , <i>Grid Computing</i> y <i>Cloud Computing</i> .	33
Tabla 4: Tabla que indica los principales servicios de <i>OpenStack</i> .	41
Tabla 5: Tabla que indica los servicios opcionales de <i>OpenStack</i> .	43
Tabla 6: Tabla que muestra las funciones básicas de la biblioteca <i>MPI</i> .	62
Tabla 7: Tabla que muestra las características generales del estándar <i>MPI</i> .	64
Tabla 8: Tabla que muestra los tiempos de cálculo para la multiplicación de matrices en <i>MPI</i> con distintos escenarios.	68

## 1. INTRODUCCIÓN

Este proyecto, se ubica en el ámbito de investigación para desarrollar un sistema computacional distribuido mediante *single-board computer* capaz de manejar datos recibidos desde distintos nodos de manera rápida y eficiente, una vez realizados los cálculos mostrar los resultados obtenidos de manera visual y fácilmente entendible para el usuario.

El propósito a largo plazo es desarrollar una estructura dentro del marco de las *Smart City*, donde cada nodo conectado a distintos sensores recogerá una serie de datos que posteriormente serán calculados obteniendo resultados para la toma de decisiones.

En la actualidad ya existen sistemas computacionales capaces de manejar grandes estructuras de datos recogidos de la ciudad, aunque estos equipos pueden proporcionar todos los recursos y potencia necesaria para mantener y administrar una *Smart City* no son totalmente eficientes, esto se debe a la falta de eficacia para llevar a cabo todas las tareas de manera escalable, ya que no siempre es necesario utilizar todo su potencial, y a veces el consumo de energía puede ser mucho más elevado de lo requerido. Esta problemática se debe a que generalmente los sistemas computacionales se encuentran centralizados, siendo necesario enviar todos los datos de la ciudad a un mismo *Data Center*.

En las ciudades existen gran cantidad de sensores que recogen información, se debe dar valor a esos datos gestionándolos para ofrecer servicios al ciudadano. Estos sensores están usualmente conectados a placas o nodos con la potencia de mini-computadoras o *single-board computer* que procesan la información.

Los nodos urbanos deben tener un microprocesador, microcontrolador, sensores y otros servicios para la ciudadanía. Este proyecto plantea el tipo de estructura a diseñar para la gestión de los nodos de las ciudades inteligentes y aprovechar el potencial computacional de estas placas para poder ejecutar otro tipo de servicio.

En este caso, el procesador utilizado para llevar a cabo el proyecto experimental es la placa *Raspberry Pi 2(RP2)*, la que se ha puesto a prueba exponiéndola a los distintos

servicios necesarios para crear un sistema distribuido capaz de gestionar, paralelizar, distribuir y calcular datos. En la parte experimental se han realizado tres pruebas distintas para comprobar las capacidades del *hardware* de las máquinas *RP2*.

- Comprobación de la capacidad de emular y virtualizar, corriendo un sistema operativo *x86* sobre *ARM*.
- Instalación de un *software* potente de distribución *cloud computing*.
- Creación de un sistema distribuido sencillo con un estándar abierto de código C. Este sistema se ha testeado mediante un programa realizado en C.

## 1.1. *Smart City*

La densidad y el tamaño de la población aumenta rápidamente cada año, esto implica un mayor consumo de electricidad y servicios en las ciudades. En consecuencia, los recursos para la producción de energía disminuyen, por este motivo cada vez más están evolucionando las nuevas formas de producir energía eficiente y sostenible, las llamadas fuentes de energías alternativas.

Las *Smart City* utilizan la tecnología para mejorar los servicios de las ciudades y reducir el consumo de recursos y costes; con toda la información que la ciudad genera debe ser capaz de predecir, gestionar y responder los desafíos locales y globales relacionados con los datos obtenidos en los sensores desplegados por la ciudad. El tipo de datos más comunes recogidos por las ciudades inteligentes son sobre: la energía, el transporte, el clima, la contaminación del aire y agua.

No existe una única definición para *Smart City* sin embargo algunas de las características más comunes que deberían tener son: el uso de sensores, capacidad de distribución de carga, es decir balancear el trabajo de cálculo necesario entre diferentes sensores, reajustamiento en caso de pérdida para evitar que el conjunto se deteriore por el fallo de uno de los sensores o nodos, capacidad de cálculo y de control para mejorar la funcionalidad general del sistema y suministro de la energía eléctrica. Debe ser un sistema flexible y escalable para hacer frente a cualquier futuro comportamiento de la población.

Las ciudades inteligentes son propensas a tener un comportamiento más dinámico en términos de servicios y aplicaciones. Los nuevos servicios deben estar disponibles



simplemente añadiendo nodos o equipos a la ciudad. Esto plantea retos evidentes para que sea lo más eficiente y sostenible posible, sobre todo teniendo en cuenta que una ciudad inteligente es probable que tenga millones de sensores y otros dispositivos. Por lo tanto, con el fin de organizar un sistema de red más eficiente, inteligente, sostenible y fiable en una ciudad, es esencial tener en cuenta y trabajar con tecnologías de la información, así como las redes virtuales entre nodos interconectados en forma de *Cloud* o *Cluster*.

## 1.2. Objetivos

Los objetivos del proyecto se centran principalmente en la implementación de un sistema computacional distribuido mediante algún tipo de *single-board computer*, estudiar y analizar las posibilidades que nos ofrece el *hardware*, y buscar un *software* que nos permita ser escalables en cualquier situación y facilidad para la gestión del sistema distribuido.

El sistema distribuido debe ser modular y escalable, y a la vez tener en cuenta el consumo de energía; aunque esté soportado por un conjunto de nodos debe ser agnóstico y aparecer como un único sistema unificado.

El proyecto ha sido realizado gradualmente y de esta forma se ha estructurado el documento. La memoria está dividida en 2 grandes bloques teóricos:

- **Hardware:** En este punto se definen básicamente los componentes físicos utilizados. Tal y como se menciona en la introducción del proyecto, existe la limitación de hardware. Utilizaremos y experimentaremos con la *SBC Raspberry Pi 2*, ya que era el *hardware* disponible para realizar el proyecto.
- **Software:** Estudio de las capacidades necesarias del *software* para la implementación final del proyecto. Mención teórica de los distintos test realizados sobre las mini-placas. Obtención del *software* capaz de gestionar grandes volúmenes de datos. Programa sencillo de ejemplo para ver las capacidades del servicio lógico elegido.

Finalmente, en el Anexo se puede encontrar el desarrollo práctico y experimental de las distintas implementaciones, para testear las características necesarias estudiadas durante la parte teórica del proyecto y así comprobar la capacidad de la *Raspberry*.

## 2. HARDWARE

La elección del *hardware* utilizado es una parte muy importante en el proceso de implementación del proyecto, aun así, estaba ligeramente limitado al uso de las placas *Single Board Computer (SCB) Raspberry Pi 2* (ver Figura 1), que eran las que tenía disponibles para crear el sistema distribuido.

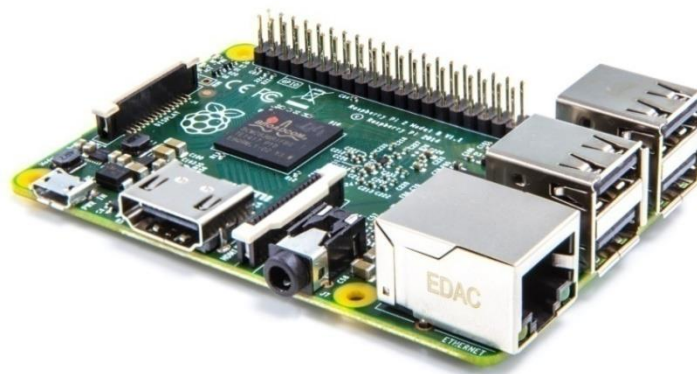


Figura 11: Imagen de la placa Single Board Computer *Raspberry Pi 2*.

El diseño del *hardware* es esencial para ahorrar tiempo y dinero cuando se trata de eficiencia energética, rápido procesamiento y consumo de energía. Se busca una alta potencia de cálculo debido a que el objetivo es implementar una combinación de varios equipos eficientes que estén conectados entre sí de manera transparente. El resultado es un conjunto de placas potentes, más económicas y eficientes que un ordenador único de potencia de cálculo similar.

Cada *Single Board computer* está formada por múltiples componentes entre ellos, la Unidad Central de Procesamiento (*CPU*) intérprete de las instrucciones del procesador. Esta es capaz de procesar millones de ejecuciones de instrucciones para que el dispositivo funcione de la manera esperada. Por lo tanto, la velocidad de cálculo y respuesta; indispensable en la experiencia del usuario, y la eficiencia energética vinculada a la vida de la batería, son muy importantes en la decisión final para la elección de la *CPU*. Como se menciona anteriormente las *SBC* de las que se disponía para realizar el proyecto eran *Raspberry Pi 2*, aun así, en las siguientes secciones veremos brevemente en una primera instancia porque este *hardware* dispone de las condiciones mencionadas para cumplir los objetivos del proyecto.

---

<sup>1</sup> [https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB\\_-comp.jpeg](https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_-comp.jpeg)

## 2.1. Arquitectura RISC / Arquitectura CISC

*Raspberry Pi 2* está diseñada sobre arquitectura *ARM*, esto nos ofrece las características necesarias para conseguir implementar el sistema distribuido deseado. A continuación, se exponen las características de las dos arquitecturas más utilizadas actualmente, para poder compararlas, y concluir que el juego de instrucciones que más se ajusta al proyecto es *ARM*<sup>2</sup>.

Las dos filosofías de diseño de las que hablamos son: las arquitecturas *CISC* y *RISC*. La arquitectura *CISC* se refiere a los microprocesadores tradicionales que operan con grupos grandes de instrucciones de procesador (lenguaje de máquina). Los microprocesadores *INTEL 80xxx* están dentro de esta categoría. Los procesadores *CISC* tienen un set de instrucciones complejas por naturaleza que requieren de varios a muchos ciclos para completarse.

La arquitectura *RISC* a diferencia de los *CISC* tiene un set de instrucciones simples requiriendo uno o pocos ciclos de ejecución. Estas instrucciones pueden ser utilizadas más eficientemente que la de los procesadores *CISC* con el diseño de *software* apropiado, resultando en operaciones más rápidas.

Dentro de cada equipo físico se encuentra la Unidad Central de Procesamiento (*CPU*). El trabajo de la *CPU* es llevar a cabo una serie de instrucciones para controlar el *hardware* instalado en el ordenador o dispositivo. Los ordenadores son una tecnología compleja y se ejecutan millones de instrucciones para hacerlos funcionar como se espera. La siguiente tabla es una comparación de las dos arquitecturas informáticas más utilizadas *ARM* y *x86*. La principal diferencia entre los dos y la característica que plantea la gran diferencia en el consumo de energía es el tipo de juego de instrucciones. *x86* funciona con una instrucción compleja fijada directamente a memoria, mientras que *ARM* utiliza un conjunto reducido de instrucciones para guardar y cargar datos de la memoria. De ahí la diferencia en el precio y el consumo de energía, sin embargo, un ordenador *x86* estándar de Intel es equivalente a 5-6 computadoras de una sola placa.

---

<sup>2</sup> [https://es.wikipedia.org/wiki/Arquitectura\\_ARM](https://es.wikipedia.org/wiki/Arquitectura_ARM)

A continuación, se muestra una tabla (ver Tabla 1) con las diferencias más relevantes de ambas arquitecturas.

<b>Arquitectura RISC</b>	<b>Arquitectura CISC</b>
Pocos ciclos de reloj por instrucción	Muchos ciclos de reloj por instrucción
Set de instrucciones simples	Incorpora set de instrucciones simples y complejas para operaciones de alto nivel
Tiempo de cálculo/ejecución bajo	Tiempo de cálculo/ejecución alto
Direccionamiento basado en operaciones de registro a registro, y operaciones de carga y recuperación de datos hacia y desde la memoria.	Direccionamiento de memoria a memoria
Muchos registros en un procesador	Pocos registros
Bajo consumo de energía	Alto consumo de energía
Altamente escalable	Poco escalable
Bajo rendimiento	Alto rendimiento
Permite virtualización	Permite virtualización
Bajo Coste	Alto coste

Tabla 1: Tabla comparativa RISC vs CISC.

Aunque la arquitectura RISC<sup>3</sup> presente componentes más potentes, su coste es realmente alto y el consumo de energía es elevado; hay que tener en cuenta que en el presente proyecto hay tres características básicas que buscamos en el *hardware*, poco consumo de energía, escalabilidad para añadir más sistemas y poder paralelizar las tareas para aprovechar al máximo la capacidad del dispositivo, y todo a un precio final económico. Con una o múltiples máquinas CISC obtendríamos un rendimiento alto, pero no llegaríamos a aprovechar toda su capacidad y el precio del conjunto sería relativamente abultado, en cambio una mejor opción para alcanzar el bajo consumo de energía, aspecto relevante en este proyecto, es mediante el uso de las computadoras de una sola placa de arquitectura ARM. Un factor importante a tener en cuenta es el bajo coste y el alto rendimiento final, ya que un ordenador x86 de gama media puede tener un precio muy elevado, en cambio con un conjunto de placas ARM tenemos el mismo rendimiento a un coste muy por debajo de la otra alternativa.

<sup>3</sup> [https://ca.wikipedia.org/wiki/Advanced\\_RISC\\_Machines](https://ca.wikipedia.org/wiki/Advanced_RISC_Machines)

El consumo medio de energía de un procesador *ARM* es de alrededor de 5W aproximadamente, la tarjeta *RPi2* consume 4W. Seis de ellas son iguales en potencia computacional a un ordenador *x86* estándar. Un estándar *x86* consume 45W y seis *Raspberry Pi 2* consumen 24W, que es casi la mitad del consumo de energía.

Un sistema distribuido compuesto por un grupo de *Raspberry Pi 2* es escalable, ya que permite conectar o desconectar las máquinas según la cantidad de datos que queremos manejar, si se necesita un alto poder computacional para realizar una tarea, en un sistema distribuido de *SBC*, se pueden administrar y añadir al sistema nuevas máquinas, por el contrario, si lo que se necesita es menos potencia las máquinas no necesarias pueden desconectarse para resolver el problema, eso permite ahorrar energía respecto a un procesador *CISC* con las mismas características que el conjunto de máquinas.

Cabe destacar que la poca complejidad de las instrucciones y el uso de pipeline permiten a los sistemas *RISC*, resolver de manera más rápida y eficiente los programas a ejecutar.

Una sola placa *SBC* con arquitectura *ARM* dispone de los mismos elementos esenciales que un ordenador de arquitectura *x86*, su tamaño reducido la dotan de un peso ligero.

Además del consumo de energía, el *hardware* puede permitir la virtualización para poder gestionar múltiples sistemas virtuales entre todos los nodos, también se ha investigado si permite emular para ver sus posibilidades, ya que nos podría interesar ejecutar otra arquitectura como *CISC x86* en alguna ocasión, servicio o aplicación. Esta observación se trata más a fondo en el apartado de 3. *Software*.

En el mercado encontramos una amplia diversidad de placas *SBC*, desde distintas fuentes se puede corroborar que la potencia, la capacidad para virtualizar y el precio de la *Raspberry Pi 2* cumplen con los requisitos establecidos.

## **2.2. Características *Raspberry Pi 2***

*Raspberry Pi 2 Modelo B* (ver Figura 2) es la segunda generación de *Raspberry Pi*, su antecesora es el modelo *Raspberry Pi 1 Model B+*. Si las comparamos el núcleo del sistema operativo de *RBP2* se ha actualizado para aprovechar al máximo la última

tecnología *ARMv7*, respecto a la *CPU ARMv6*, de esta manera se puede ejecutar toda la gama de distribuciones *ARM GNU / Linux* este punto representa un gran aumento de rendimiento con respecto a sus antecesores basándose en un procesador *Cortex-A7* de cuatro núcleos. Aun así, se ha mantenido la compatibilidad con versiones anteriores de *hardware* y *software* con la *Raspberry Pi 1 modelo A+/B+*, por lo tanto, puedes utilizar todos los accesorios y expansiones disponibles.

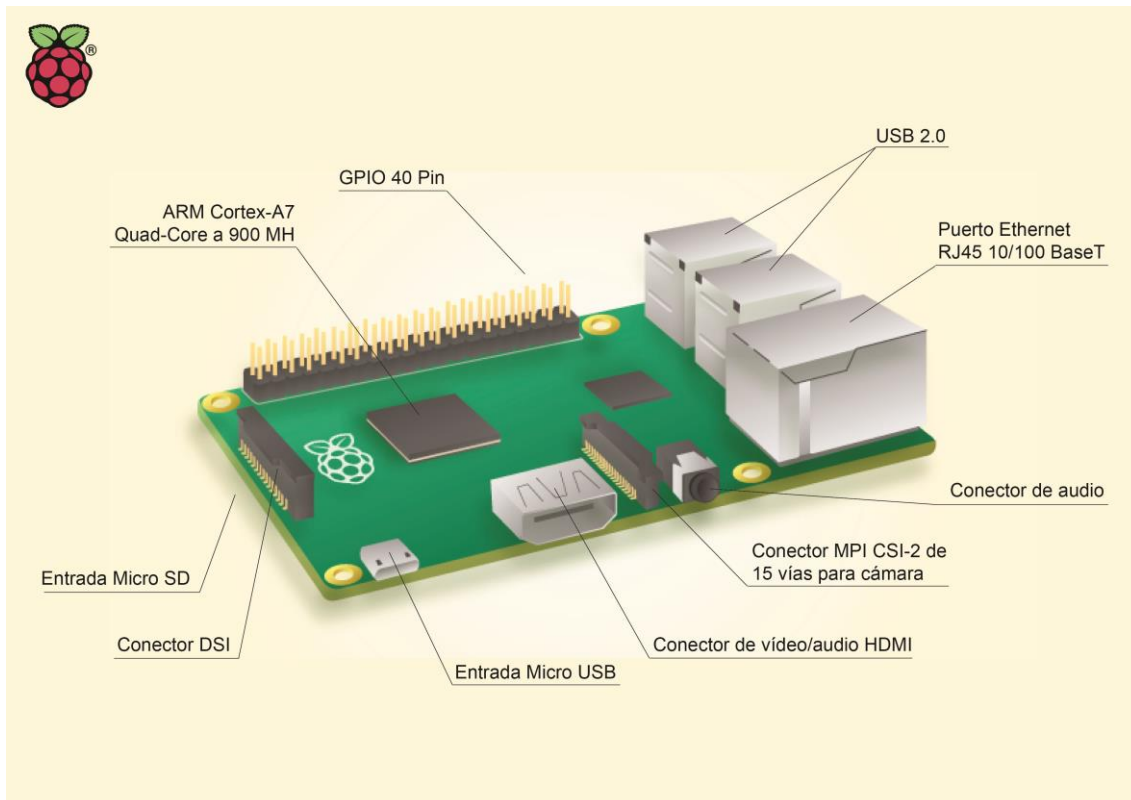


Figura 2: Infografía donde se indican las partes esenciales de la *Raspberry Pi 2*.

La *Raspberry Pi 2 model B* ahora también ofrece *1GB* de memoria *RAM* para las aplicaciones con más requisitos de memoria o cálculo, mientras que su antecesora disponía solo de *512MB*, la mitad. Aunque se hayan realizado mejoras de una máquina respecto a la otra, el precio se ha mantenido igual en ambas.

El precio también es una característica importante a destacar ya que es relativamente bajo por lo que puede llegar a ofrecer el hardware; existen muchas posibilidades de uso en el mercado; sistema operativos basados para la implementación de un centro multimedia, emulador de videoconsolas antiguas, proyecto para *Smart TV*, etc.

Además, *Raspberry Pi* arranca el sistema desde una imagen preinstalada en una tarjeta *microSD* y en apenas 15 segundos.

Para más detalle, a continuación, se presenta una tabla (ver Tabla 2) con las especificaciones técnicas<sup>4</sup> de *Raspberry Pi 2 model B*:

<i>RASPBERRY PI 2 MODELO B</i>	
<b>Precio</b>	35€
<b>System-on-chip (SoC)</b>	Broadcom BCM2836
<b>Peso</b>	45g
<b>Peso de la batería de litio</b>	5g
<b>Dimensiones</b>	86x57x20mm
<b>Consumo energético</b>	800mA (4,0W)
<b>Procesador/CPU</b>	ARM, Cortex-A7 Quad-Core a 900MHz
<b>Instrucciones</b>	ARMv7 32bits
<b>Tecnología de la memoria</b>	LPDDR2-SDRAM 1GB
<b>Tipo de memoria del ordenador</b>	DIMM
<b>Coprocessador grafico/GPU</b>	GPU 250MHz VideoCoreIV
<b>Puertos USB</b>	4 conectores hembra USB 2.0
<b>Puertos Micro USB</b>	1
<b>Puertos Ethernet</b>	1 Conector hembra Ethernet RJ45 10/100 BaseT
<b>Puerto Full HDMI</b>	Si
Conector MPI CSI-2 de 15 vías para cámara de video HD Raspberry Pi	
Conector de interfaz serie display 15 vías	
Conector Macho de 40 pines para buses serie y GPIO(compatible con el conector macho de 26 pines Raspberry Pi 1)	
<b>Voltaje</b>	5 voltios a 2ª a través de conector hembra microUSB
<b>Almacenamiento integrado</b>	MicroSD

<sup>4</sup> <http://www.xatakahome.com/trucos-y-bricolaje-smart/probamos-la-nueva-raspberry-pi-2-a-fondo>

<b>Lector de tarjetas</b>	MicroSD
<b>Entrada de Video</b>	Conector MIPI CSI (permite instalar un modulo de cámara desarrollado por la RPF)
<b>Salida de Video</b>	HD 1080p Salida de video compuesto (PAL/NTSC) Conector hembra de video/audio HDMI 1.3 y 1.4 Conector hembra de salida de video compuesto/audio de 3,5mm 4 polos
<b>Interfaz de la cámara</b>	CSI
<b>Interfaz de pantalla</b>	DSI

Tabla 2: Tabla con especificaciones técnicas de la *Raspberry Pi 2 model B*.

## 2.3. Hub, Switch y Router

Para poder implementar un sistema distribuido necesitamos que cada *Single Board Computer* pueda comunicarse mediante el envío de paquetes o mensajes con las demás placas, para ello veremos brevemente la descripción de 3 dispositivos de red capaces de cubrir esa necesidad, y cuál de ellos es más conveniente para el conjunto.<sup>5</sup>

### HUB

El *hub* (concentrador) es el dispositivo de conexión más básico. Es utilizado en redes locales con un número muy limitado de máquinas. No es más que una toma múltiple *RJ45* que amplifica la señal de la red (base 10/100).

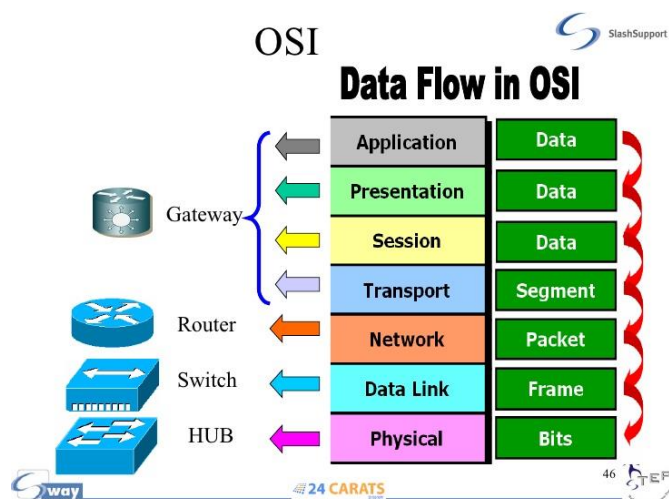


Figura 3: Infografía explicativa de la estructura en capas del modelo OSI.

En este caso, una solicitud destinada a un servidor de la red será enviada a todas las

<sup>5</sup> <http://es.slideshare.net/williamflazh/definiciones-sobre-switch-router-modem-wimax-acces-point>



máquinas. Esto reduce de manera considerable el ancho de banda y ocasiona problemas de comunicación en la red.

Los *hubs* trabajan en la primera capa del modelo *OSI* encargada de la parte física del sistema, como vemos en la imagen (ver Figura 3).

## SWITCH

Un *Switch* es un dispositivo de interconexión utilizado para conectar equipos en red formando lo que se conoce como una red de área local (*LAN*) y cuyas especificaciones técnicas siguen el estándar conocido como *Ethernet*.

Este dispositivo trabaja en conjunción con las dos primeras capas del modelo *OSI*, es decir que a diferencia del dispositivo *hub*, trabaja en la capa de enlace de datos donde a partir de la tarjeta de red distribuye los datos a cada máquina de destino, mientras que el *hub* envía todos los datos a todas las máquinas que responden.

La función básica de un *switch* es la de unir o conectar dispositivos en red (ver Figura 4). Es importante tener claro que un *switch* no proporciona por si solo conectividad con otras redes, y obviamente, tampoco proporciona conectividad con Internet. Para ello es necesario un *router*.

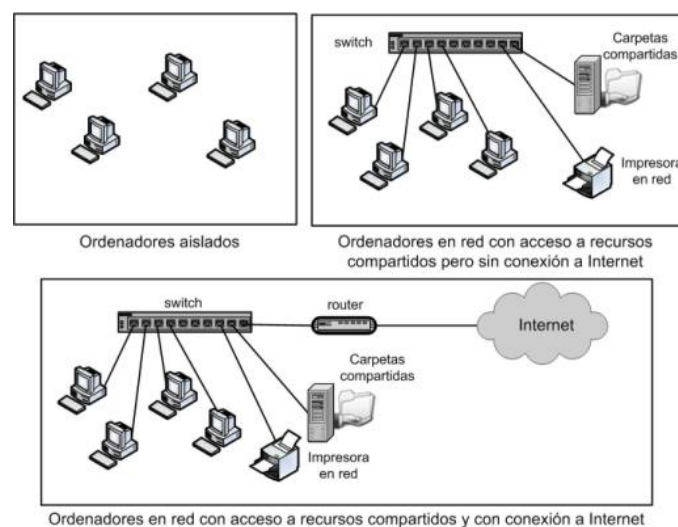


Figura 4: Infografía explicativa del funcionamiento de un *switch* y un *router*.

La función básica que realiza un *switch* se conoce como conmutación y consiste en transferir datos entre los diferentes dispositivos de la red. Para ello, los *switches* procesan la información contenida en las cabeceras de las tramas *Ethernet*.

Sin entrar mucho en detalle en el funcionamiento de las tramas *Ethernet* podemos decir que son una tecnología de transmisión de datos para redes locales cableadas que divide los datos que se tienen que transmitir en tramas y a cada trama se le añade una determinada información de control llamada cabecera. Dicha cabecera contiene la dirección *MAC* tanto del emisor como del receptor. Los *switches* guardan en una tabla las direcciones *MAC* de todos los dispositivos conectados junto con el puerto en el que están conectados, de forma que cuando llega una trama al *switch*, dicha trama se envía al puerto correspondiente.

Concebido para trabajar en redes con una cantidad de máquinas ligeramente más elevado que el *hub*, éste elimina las eventuales colisiones de paquetes (una colisión aparece cuando una máquina intenta comunicarse con una segunda mientras que otra ya está en comunicación con ésta..., la primera lo reintentará luego).

## ROUTER

El *router* trabaja permitiendo conectar múltiples redes y enviar paquetes destinados ya sea a sus propias redes o a otras redes, es decir tiene la capacidad de distribuir paquetes *IP* (ver Figura 5).

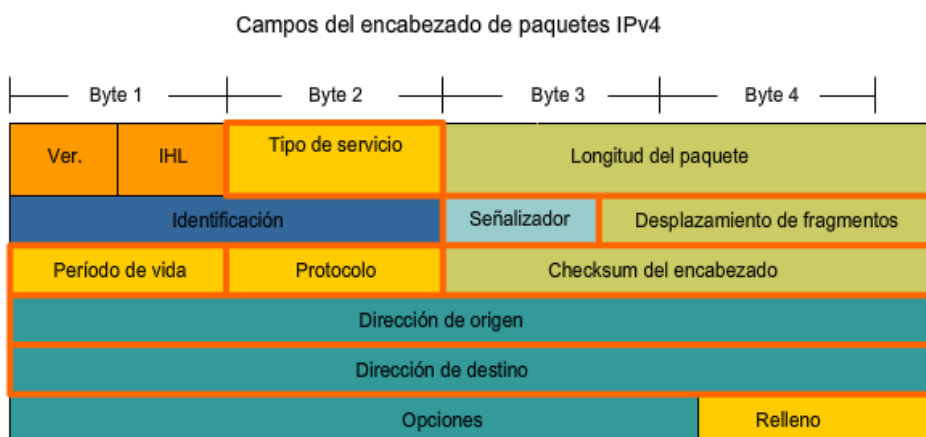


Figura 5<sup>6</sup>: Infografía informativa de los campos que forman el paquete IP.

<sup>6</sup> <http://i1.wp.com/guzman6001.byethost10.com/wp-content/uploads/2013/02/campos-del-encabezado-ipv4.png>

Es capaz de diseccionar los paquetes para obtener la *IP* de destino del paquete. Cuando el *router* recibe un paquete de información lo primero que hace es revisar su tabla de enrutamiento para verificar si la *IP* de destino coincide con alguna de las *IP* a las que está asignado cuando se encuentra una coincidencia, el paquete se encapsula en la trama de enlace de datos de la Capa 2 para esa interfaz de salida. El tipo de encapsulación de enlace de datos depende del tipo de interfaz, como por ejemplo *Ethernet* o *HDLC*, de lo contrario mandará la información a un siguiente *router* para hacer el mismo proceso y así encontrar la dirección *IP* destino seleccionada (ver Figura 6). Su ubicación en el modelo *OSI* es la capa 3, la capa de red donde puede gestionar la distribución de los paquetes a partir de la *IP*.

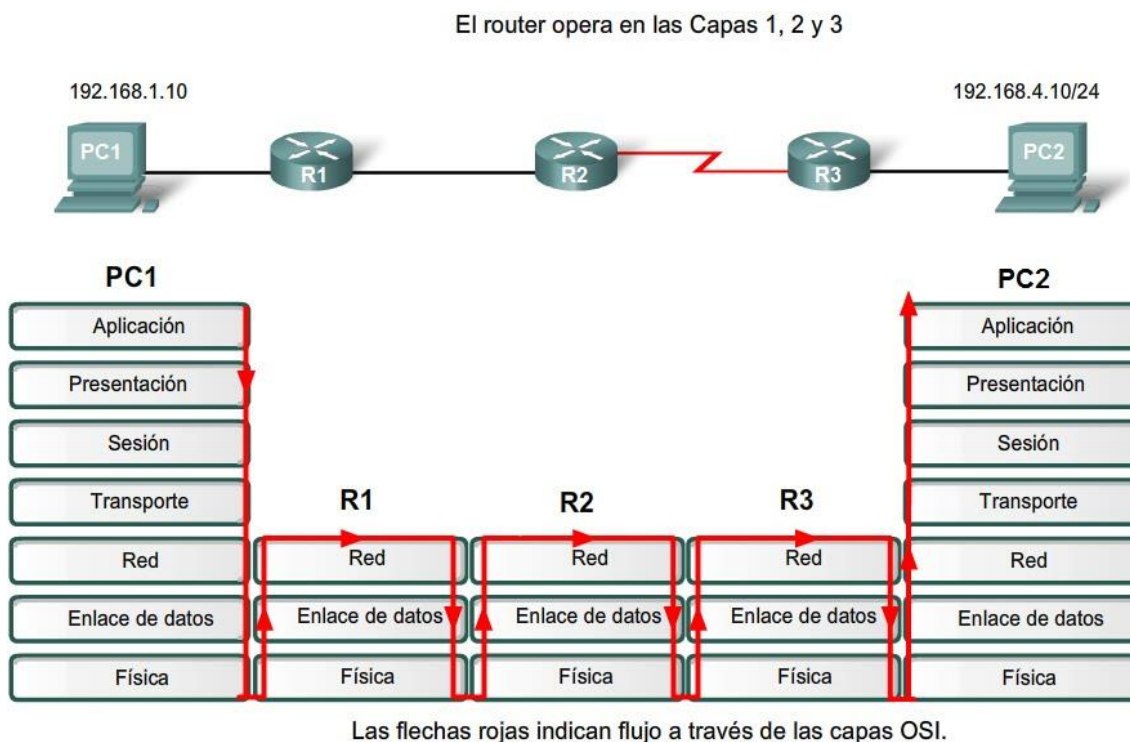


Figura 6<sup>7</sup>: Infografía explicativa del flujo de envío de paquetes por parte del *router*.

El *router* se utiliza en instalaciones más grandes, donde es necesaria (especialmente por razones de seguridad y simplicidad) la creación de varias sub redes. El *router* equivale a un *PC* gestionando varias conexiones de red. Estos dispositivos son compatibles con *NAT*, lo que permite utilizarlos en redes más o menos extensas disponiendo de gran cantidad de máquinas y poder crear sub redes. También tienen la función de cortafuegos (*firewall*) para proteger la instalación.

<sup>7</sup> [https://sites.google.com/site/redescna2cisco/\\_/rsrc/1342855927655/envio-de-paquetes/Nueva%20imagen%20\(6\).bmp](https://sites.google.com/site/redescna2cisco/_/rsrc/1342855927655/envio-de-paquetes/Nueva%20imagen%20(6).bmp)

Un *router* envía paquetes entre redes mediante el cálculo de la mejor ruta y la lectura de las direcciones *IP*, proceso en la capa de red. Para concluir un *router* utiliza *WAN*, *Wide Area Network*, lo que significa que también permite conectar redes *LAN*, *Local Area Network*.

### 2.3.1. Decisión

Después de examinar los 3 dispositivos, y pensando en un escenario para *Smart City*, vinculando un conjunto de nodos distribuidos y compuestos por varias placas por la ciudad, se concluyó que el mejor dispositivo de comunicación a utilizar es el *router*. Los principales motivos para tal decisión se basaron en las siguientes observaciones: permite conexiones *LAN* y *WAN*, se puede implementar una alta seguridad, permite la adhesión de otros *routers* al sistema distribuido, y finalmente, es capaz de distribuir los mensajes y paquetes de manera rápida y eficiente calculando la mejor ruta posible mediante la tabla de direcciones *IP*.

## 3. SOFTWARE

Llegado al presente apartado, ya tenemos los elementos *hardware* que vamos a utilizar, las placas *SBC Raspberry* como nodos operacionales del sistema distribuido y los *routers* como intercomunicadores de las máquinas. Ahora necesitamos encontrar el sistema lógico capaz de conseguir cubrir las necesidades mencionadas anteriormente como: velocidad de cálculo, virtualización/emulación, escalabilidad para añadir nuevos nodos, y distribución flexible de la capacidad operacional de las *Raspberrys*.

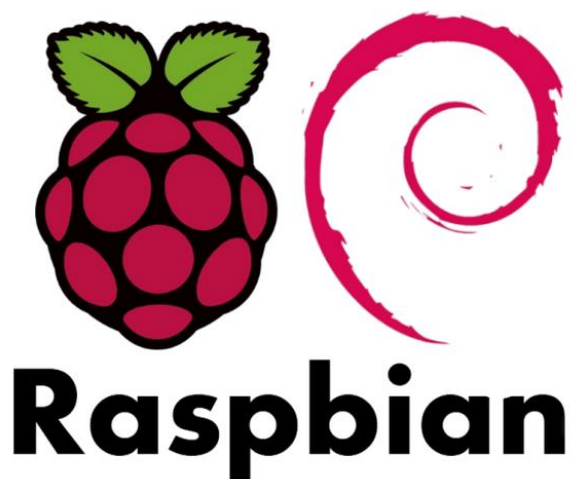
En primer lugar, se ha hecho un breve análisis para elegir el sistema operativo que, básicamente, permite al resto de los programas funcionar adecuadamente, facilitando también la interacción entre los componentes físicos y el resto de las aplicaciones.

Para entender un poco el funcionamiento del sistema distribuido y la capacidad necesaria del sistema operativo, podemos volver al escenario de las *Smart City* donde tendremos miles de placas interconectadas entre ellas mediante *routers*, y recogiendo datos con distintos sensores. Posteriormente estos datos serán procesados por los nodos correspondientes para hacer útil toda la información acumulada. Esa información ya precalculada será recogida por un servidor central para finalmente

mostrarla a los usuarios vía web.

Por lo tanto, buscamos básicamente un sistema operativo optimizado especialmente para la placa que vamos a utilizar, no es requerido que disponga de interfaz gráfica, de esta forma el sistema será más ligero, debe permitir obtener datos de sensores y realizar los cálculos de manera eficaz, rápida y distribuida.

Aunque existen diversos Sistemas Operativos (SO) optimizados para la *Raspberrys*, la elección de este ha sido la versión oficial *Raspbian*. Los motivos de tal elección son los siguientes: es un sistema operativo libre basado en *Debian Wheezy* optimizado para el *hardware* de *Raspberry Pi*, al ser una distribución de *GNU/Linux* esta soportado por la comunidad y permite



posibilidades infinitas sobre el control de la máquina. Destaca también el menú "*raspi-config*" que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente.

La versión utilizada para el proceso experimental del proyecto ha sido *Raspbian Jessie Lite* versión minimalista, pero con las capacidades necesarias para el planteamiento del proyecto.

Los siguientes apartados recogen tres test experimentales distintos sobre las capacidades de la *Raspberry*. El primero pretende comprobar la flexibilidad de las placas *SBC* para virtualizar/emular; los otros dos engloban la teoría para decidir que tecnología utilizar y las implementaciones experimentales para conseguir un sistema distribuido en versión reducida.

### 3.1. Emulación y virtualización

Como se ha menciona en puntos anteriores del proyecto las placas utilizadas tienen la capacidad para virtualizar, aun así, en este punto hemos querido experimentar sobre la capacidad para emular una arquitectura *x86* con *ARM*.

Un emulador es un programa de *software* que simula la funcionalidad de otro programa o componente de *hardware*, es decir puede simular una máquina al completo, el *hardware* incluido. Comúnmente es utilizado para desarrollar *software* sin que el *hardware* esté disponible físicamente. Dado que implementa funcionalidad por *software*, proporciona una gran flexibilidad y la capacidad de recopilar información muy detallada acerca de la ejecución.

El inconveniente de la emulación es que, al compartirse el *hardware* de la máquina, el rendimiento general de esta baja considerablemente.

Con la virtualización<sup>8</sup>, el programa huésped se ejecuta realmente en el *hardware* subyacente. El *software* sólo media entre los accesos de las diferentes máquinas virtuales y el *hardware* real. Así, éstas son independientes, y pueden ejecutar programas a velocidad casi nativa.

La virtualización permite que múltiples instancias de sistemas operativos puedan ejecutarse concurrentemente en un solo computador de una manera dinámica y compartiendo los recursos físicos disponibles, tales como *CPU*, almacenamiento, memoria y los dispositivos de entrada y salida *I/O*.

*Raspberry Pi 2* pertenece a la familia de microprocesadores *RISC*, *Advanced RISC Machines(ARM)*, el primer objetivo es comprobar la capacidad de emular una arquitectura *x86* en estas placas, ya que *x86* y *ARM* son arquitecturas distintas como se explica a continuación, además esta funcionalidad puede ser necesaria en según qué ambientes en una *Smart City*.

### **3.1.1. Primera implementación práctica: Emulación x86 sobre ARM**

Para emular el comportamiento de una arquitectura *x86* sobre la arquitectura *ARM* de las placas se ha utilizado el emulador *Qemu*; aunque también se ha visto otro emulador, *Exagear*, que hasta que no apareció el modelo *RPI2* con procesador *ARMv7 Cortex-A7* no era una opción, debido a que el procesador de la Primera

---

<sup>8</sup> <http://www.welivesecurity.com/la-es/2014/07/28/virtualizacion-o-emulacion-esa-es-la-cuestion/>

generación de *Raspberry* es *ARMv6* y no es compatible con *Exagear*. Aunque la información obtenida muestra que es un *software* fácil de usar con un funcionamiento transparente y con estadísticas más altas en todos los aspectos, se ha descartado debido a que no es de libre distribución, sino que pertenece a la empresa *ELTECHS*<sup>9</sup>.

*Qemu*<sup>10</sup> es un emulador de procesadores basado en la traducción dinámica de binarios (conversión del código binario de la arquitectura fuente en código entendible por la arquitectura huésped). *Qemu* también tiene capacidades de virtualización dentro de un sistema operativo, ya sea *GNU/Linux* o *Windows*. Este emulador puede ejecutarse en cualquier tipo de microprocesador o arquitectura (*x86*, *x86-64*, *PowerPC*, *MIPS*, *SPARC*, etc.).

El objetivo principal es emular un sistema operativo dentro de otro sin tener que reparticionar el disco duro, empleando para su ubicación cualquier directorio dentro de éste.

El programa no dispone de interfaz gráfica (*GUI*) pero en este caso no es necesario ya que trabajamos con la versión minimalista del sistema operativo y no utilizamos ningún tipo de escritorio gráfico.

*Qemu* puede operar de dos maneras distintas: emular un proceso, o emular un sistema informático completo, incluyendo procesador y varios periféricos.



En el primer test experimental se ha testeado la capacidad para emular un sistema *x86*, concretamente *Windows 98*

En el Anexo A podemos encontrar el Manual completo sobre esta prueba, además al ser el primero de las tres pruebas que se han hecho se incluye, al principio del documento, una guía de como instalar y configurar el sistema operativo *Raspbian* en *Raspberry*.

---

<sup>9</sup> <https://eltechs.com/product/exagear-desktop/>  
<http://hackerboards.com/emulator-brings-x86-linux-apps-to-arm-devices/>

<sup>10</sup> [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page)  
<http://mogaal.com/articulos/qemu.html>  
<https://launchpad.net/ubuntu/trusty/+package/qemu-system-x86>

### 3.1.2. Resultados

La parte experimental de esta primera prueba se ha realizado sobre una única máquina *Raspberry Pi 2*, el test ha sido positivo ya que se ha conseguido el objetivo de emular *Windows 98* en la *SBC*. Aun así, los resultados no han sido del todo esperanzadores, el sistema operativo inquilino se ha instalado correctamente y ha detectado los dispositivos de entrada y salida, pero se hacía prácticamente imposible trabajar con el equipo debido al alto consumo de recursos, derivando a un bajo rendimiento y poca velocidad, en resumen, una eficacia de trabajo nula.

Podemos concluir que *Raspberry* puede emular; aunque el consumo de memoria es muy elevado y por lo tanto no sería una opción que interese para los objetivos finales de un sistema distribuido rápido y eficiente. Es posible que con el *software Exagear* se mejoren los tiempos, en ese caso se debería comprobar, si es así, sería interesante contemplarlo para un futuro.

## 3.2. Modelos de Sistema Distribuido

El planteamiento del proyecto enfocado a las *Smart City* evoca a implementar un sistema distribuido capaz de gestionar múltiples datos, procesarlos y ofrecer servicios al usuario final. El concepto usuario final se puede interpretar de dos maneras distintas. El usuario puede ser cualquier ciudadano que des de su casa vía web pueda utilizar los servicios o datos, o bien que los usuarios sean otras máquinas que, según los resultados de los datos recogidos y calculados por los nodos, generen estadísticas y actúen en consecuencia. En ese caso el servicio se ofrecería a la población de manera dinámica sin que los ciudadanos intervinieran. En cualquier caso, la cardinalidad, la topología y la estructura general del sistema serán transparentes al usuario.

Todos los nodos deben de relacionarse mediante un sistema distribuido para compartir los datos y la carga de la gestión. La computación distribuida es un tipo de computación segmentada o paralela en que diferentes partes de un programa se ejecutan simultáneamente en dos o más procesadores que forman parte del mismo



sistema.

En un sistema distribuido un solo problema se descompone en varias tareas en las que cada tarea se calcula en las máquinas individuales que forman parte del conjunto. Todos los ordenadores conectados en una red distribuida se comunican entre sí para alcanzar un objetivo común, haciendo uso de su propia memoria local.

Para construir un sistema distribuido existen 3 tipos de estructura, conocidos como *Grid*, *Clúster* y *Cloud Computing*. Veremos un poco en que se basa cada una y sus características principales.

### GRID

La computación *grid*<sup>11</sup> permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado. En esta topología de computación distribuida, los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores, *clústers*...) y se encuentran conectados a través de Internet.

El término *grid* se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Puesto que la colaboración entre instituciones envuelve un intercambio de datos, y de tiempo de computación, el propósito del *grid* es facilitar la integración de todos los recursos computacionales.

Llamamos *grid* al sistema de computación distribuido que permite compartir recursos no centrados geográficamente para resolver problemas de gran escala. Los recursos compartidos pueden ser ordenadores (*PC*, estaciones de trabajo, supercomputadoras, *PDA*, portátiles, móviles, etc), *software*, datos e información, instrumentos especiales (radio, telescopios, etc.).

La computación *grid* ofrece muchas ventajas frente a otras tecnologías alternativas. La potencia que ofrecen multitud de computadores conectados en red usando *grid* es

---

<sup>11</sup> <http://gridcpm.blogspot.com.es/>  
<http://stackoverflow.com/questions/9723040/what-is-the-difference-between-cloud-grid-and-cluster>

prácticamente ilimitada, además de que ofrece una perfecta integración de sistemas y dispositivos heterogéneos, por lo que las conexiones entre diferentes máquinas no generarán ningún problema. Se trata de una solución altamente escalable, potente y flexible, ya que evita problemas de falta de recursos (cuellos de botella) y nunca queda obsoleta, debido a la posibilidad de modificar el número y características de sus componentes.

Así pues, su objetivo será el de compartir una serie de recursos en red de manera uniforme, segura, transparente, eficiente y fiable, ofreciendo un único punto de acceso a un conjunto de recursos distribuidos geográficamente en diferentes dominios de administración.

### CLUSTER

Un *clúster*<sup>12</sup> es un conjunto de ordenadores paralelos o distribuidos que se encuentran comúnmente, pero no siempre, conectados entre sí a través de redes locales. El grupo trabaja en conjunto en la ejecución de tareas de cálculo de grandes volúmenes de datos que no sería factible ejecutar en un solo ordenador. Las agrupaciones se utilizan principalmente para la alta disponibilidad, balanceo de carga y para propósitos de cálculo.

Normalmente el conjunto de dispositivos utilizados como nodos son máquinas de *hardware* similar, algo que en *grid* o *cloud* tengan, posiblemente, diferentes configuraciones de *hardware*.

En computación *clúster* se utiliza el propósito de la alta disponibilidad ya que se mantienen los nodos redundantes que se utilizan para proporcionar servicio cuando los componentes del sistema fallan. El rendimiento del sistema se mejora aquí, porque incluso si un nodo falla, hay otro nodo en espera que llevará a cabo la tarea y eliminará los puntos de falla sin ningún obstáculo. En un *clúster* hay que organizar todos los nodos para que trabajen juntos, y proporcionar la consistencia de los elementos tales como la memoria caché.

---

<sup>12</sup> [http://www.garudaindia.in/html/pdf/ggoa\\_2011/day%201/cluster\\_grid\\_cloud\\_concepts.pdf](http://www.garudaindia.in/html/pdf/ggoa_2011/day%201/cluster_grid_cloud_concepts.pdf)

El conjunto de máquinas que trabajan en un *clúster* comparten la carga de trabajo computacional y para el usuario final el conjunto es realmente como un único ordenador. El programa a ejecutar se distribuye entre todos los equipos independientes para formar un *clúster*. Esto se traduce en un trabajo computacional equilibrado entre las diferentes máquinas, mejorando el rendimiento de los sistemas.

Existe distintas ramas o grandes aplicaciones en que se utiliza un sistema distribuido *clúster*, como simulaciones de choques de automóviles, procesamiento de imágenes, aerodinámica, astrofísica, etc. y hay que destacar la minería de datos que implica tanto la obtención como el cálculo, capacidad que necesitamos para el desarrollo de los nodos para la *Smart City*.

## CLOUD

*Cloud*<sup>13</sup> es un tipo de sistema paralelo y distribuido que consiste en un conjunto de ordenadores interconectados y virtualizados que se presentan como uno o más recursos de computación unificados basados en un acuerdo de nivel de servicio de forma dinámica. La virtualización en el *cloud* es muy importante tanto para los usuarios como para el almacenamiento de datos. La computación *cloud* ofrece todo tipo de recursos tecnológicos como, *software* específico, sistemas operativos, plataforma de *hardware* u otros recursos de red virtualizados. Concluyendo se puede afirmar que se maneja, gestiona y arbitra los cuatro recursos principales de una computadora (CPU, memoria, dispositivos periféricos y conexiones de red) y así poder repartir dinámicamente dichos recursos entre todas las máquinas y recursos virtuales definidos en los distintos servidores distribuidos.

La computación *cloud* o nube es un modelo que permite tener, acceso ubicuo conveniente, es decir en cualquier lugar, a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que puedan ser rápidamente aprovisionados o liberados según la necesidad.

Este método de computación distribuida se realiza mediante la puesta en común de

---

<sup>13</sup><http://www.ijarccce.com/upload/2014/march/IJARCCCE9B%20%20a%20%20anjan%20A%20Comparative%20Analysis%20Grid%20Cluster%20and%20Cloud%20Computing.pdf>

todos los recursos de los servidores juntos, gestionados por el *software* para ofrecer servicios. Los servicios que se solicitan en una nube no se limitan al uso de aplicaciones web, también pueden ser tareas de gestión de la tecnología como la solicitud sobre los sistemas, de *software* o un *hardware* específico.

Con la ayuda de la computación *cloud*, los usuarios no necesitan comprar la infraestructura física y gastar dinero en su mantenimiento. Pueden utilizar cualquier tecnología según sus necesidades.

En conclusión, la computación *cloud* se puede definir o concebir como un modelo de servicios escalables bajo demanda para la asignación y el consumo de recursos de cómputo. Todo esto englobando el uso de infraestructuras, aplicaciones, información y un conjunto de servicios integrados para las reservas de recursos de computación, redes, datos y capacidad de almacenamiento. Y asumiendo también que estos elementos pueden construirse, abastecerse, implementarse y liberarse rápidamente, con un pequeño esfuerzo de desarrollo, control e interacción por parte del proveedor de *Cloud Computing*, en propósito de satisfacer las necesidades actuales del cliente, no siendo consciente de donde está obteniendo estos recursos y servicios.

*Cloud computing* proporciona a los desarrolladores y departamentos de Tecnologías de la Información la capacidad de concentrarse en lo que más importa y evitar arduas tareas como el aprovisionamiento, el mantenimiento y la planificación de capacidad. A medida que ha incrementado la popularidad del *cloud computing*, se han desarrollado varios modelos y estrategias de implementación para satisfacer las necesidades de los distintos usuarios.

La prestación de servicios de *cloud computing*<sup>14</sup> puede asociarse a tres modelos de negocio específicos:

***Infrastructure as a Service (IaaS)***: Este Modelo (ver Figura 7) de negocio ofrece al consumidor la provisión de procesamiento, almacenamiento, redes y cualquier otro recurso de cómputo necesario para poder instalar *software*, incluyendo el sistema operativo y aplicaciones. Ejemplo: *OpenStack*.

---

<sup>14</sup> <https://aws.amazon.com/es/types-of-cloud-computing/>

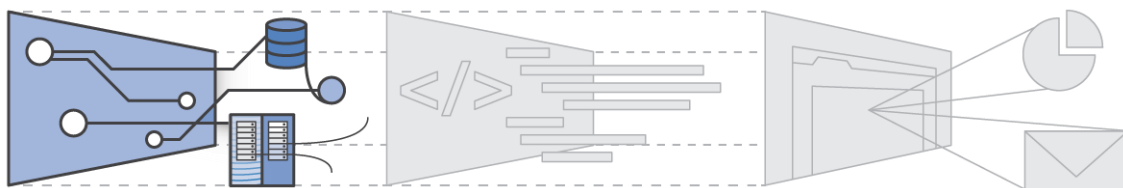


Figura 7: Figura ilustrativa del modelo *Infrastructure as a Service*

**Platform as a Service (PaaS):** Este Modelo (ver Figura 8) de negocio ofrece al consumidor (usuario) la capacidad de ejecutar aplicaciones desarrolladas por él o contratadas a terceros, a partir de los lenguajes de programación e interfaces provistas por el proveedor de servicio.

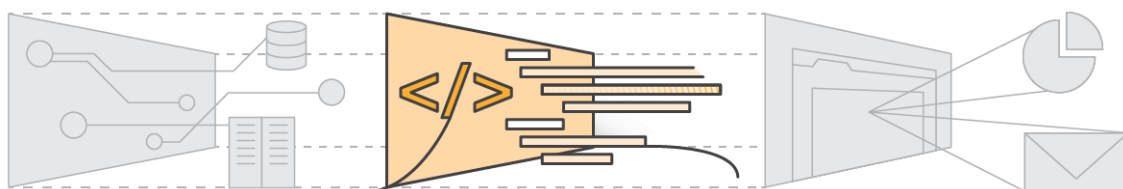


Figura 8: Figura ilustrativa del modelo *Platform as a Service*

**Software as a Service (SaaS):** Este Modelo (ver Figura 9) de negocio ofrece al consumidor (usuario) la capacidad de utilizar las aplicaciones del proveedor que se ejecutan sobre la infraestructura en la nube. Se puede acceder a las aplicaciones desde los dispositivos del cliente a través de interfaces, por ejemplo, un navegador web.

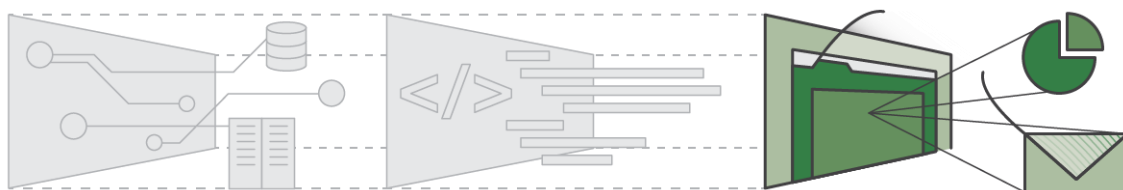


Figura 9: Figura ilustrativa del modelo *Software as a Service*

La prestación de servicios de *cloud computing* puede asociarse a tres modelos de implementación específicos:

**Nube Pública:** este Modelo de implementación del *cloud* permite que la infraestructura y los recursos lógicos que forman parte del entorno se encuentren disponibles para el público en general o un amplio grupo de usuarios. Suele ser propiedad de un

proveedor que gestiona la infraestructura y los servicios ofrecidos. Ejemplo: Servicio de *GoogleApps*.

**Nube Privada:** este Modelo de implementación del *cloud* permite que la infraestructura se gestione únicamente por una organización. La administración de aplicaciones y servicios puede estar a cargo de la misma organización o de un tercero. La infraestructura asociada puede estar dentro de la organización o fuera de ella. Ejemplo: Cualquier servicio de nube propio de la organización o contratado a un proveedor, pero cuyos recursos sean exclusivos para dicha organización.

**Nube Comunitaria:** este Modelo de implementación del *cloud* permite que la infraestructura sea compartida por diversas organizaciones y su principal objetivo es soportar a una comunidad específica que posea un conjunto de preocupaciones similares (misión, requisitos de seguridad o de cumplimiento normativo, etc.). Al igual que la Nube Privada puede ser gestionada por las organizaciones o bien por un tercero y la infraestructura puede estar en las instalaciones propias o fuera de ellas. Ejemplo: El servicio proveído por *www.apps.gov* del gobierno de *EEUU*, el cual provee servicios de cloud computing a las dependencias gubernamentales.

**Nube Híbrida:** este Modelo de implementación del *cloud* permite que se combinen dos o más tipos de Nubes anteriores, manteniéndolas como entidades separadas pero que unidas por tecnologías estandarizadas o propietarias, permitan la portabilidad de los datos y aplicaciones gestionadas.

A modo de resumen del apartado 3.2. Modelos de Sistema Distribuido, se ha elaborado una tabla comparativa<sup>15</sup> (ver Tabla 3) que dispone de las principales características de los tres tipos de distribuciones, mencionadas anteriormente:

<b>CLÚSTER COMPUTING</b>	<b>GRID COMPUTING</b>	<b>CLOUD COMPUTING</b>
Un conjunto de ordenadores similares (o idénticos) están conectados de forma local para operar como un único	Los ordenadores no tienen que estar en la misma ubicación física y pueden funcionar de forma independiente. Los nodos	Los ordenadores no tienen que estar en la misma ubicación física. Se trabaja a menudo con virtualización.

<sup>15</sup> <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.465.8919&rep=rep1&type=pdf>

equipo	pueden no estar directamente relacionados en la misma tarea.	
Los equipos de la red ejecutan el mismo sistema operativo y disponen de un <i>hardware</i> similar.	Los equipos que forman parte de una red pueden ejecutar diferentes sistemas operativos y tienen un <i>hardware</i> diferente.	La memoria, dispositivo de almacenamiento y comunicación de la red son gestionados por el sistema operativo. <i>Software</i> de código abierto como <i>Linux</i> puede apoyar la gestión de la unidad y la virtualización de la computación física básica.
Todo el sistema se comporta como un único ordenador a vistas del usuario. Los recursos deben ser administrados propiamente.	Cada nodo es autónomo, tiene su propio gestor de recursos y se comporta como una entidad independiente	Cada nodo actúa como una entidad independiente
Los equipos del <i>clúster</i> normalmente se encuentran en un solo lugar o complejo.	La red se distribuye inherentemente por su naturaleza a través de <i>LAN</i> , <i>WAN</i> o metropolitana	Las nubes se distribuyen principalmente sobre redes <i>WAN</i> .
Más de 2 equipos están conectados a resolver un problema.	Un gran proyecto se divide entre varios ordenadores para hacer uso de sus recursos	Permite múltiples aplicaciones más pequeñas para ejecutar al mismo tiempo.
Áreas de la computación: 1. Recursos Educativo 2. Sector comercial e industrial 3. Investigaciones Médicas	Áreas de la computación: 1. Predicción de Modelados y Simulaciones. 2. Ingeniería de automatización y diseño 3. Exploración de recursos energéticos	Áreas de la computación: 1. Área Bancaria 2. Área de Seguros 3. Predicción del tiempo 4. Exploración Espacial 5. Software como servicio 7. <i>IaaS (Infraestructura como servicio)</i> y <i>PaaS</i>

	4. Medicina, Área militar	
Computadoras básicas	Ordenadores de gama alta.	Un <i>hypervisor (VM)</i> en la que múltiples sistemas operativos puedan ejecutarse.
Propiedad individual	Propiedad múltiple	Propiedad individual
Baja latencia, alto ancho de banda.	Alta latencia, bajo ancho de banda	Baja latencia, alto ancho de banda.
Privacidad por contraseña a un usuario, los permisos dependen de los privilegios asignados.	Par de claves de autenticación publica/privada, asignación de usuario a una cuenta.	Cada usuario / aplicación está provista de una máquina virtual. Alta seguridad / privacidad garantizada. Apoyo para el establecimiento del control de acceso por archivo.
Gestión de usuarios está centralizada.	Gestión de usuarios está descentralizada.	Gestión de usuarios es centralizada o descentralizada, depende del servicio.
Gestión de recursos centralizada	Gestión de los recursos se distribuye.	Gestión de recursos centralizada / distribuida
En caso de fallo de tarea esta se reinicia. En caso de fallo de la máquina las otras asumen la carga.	En caso de fallo de tarea esta se reinicia. En caso de fallo de la máquina las otras asumen la carga.	Un fuerte apoyo para la conmutación por error y la replicación de contenido. Las máquinas virtuales se pueden migrar fácilmente de un nodo a otro.

Tabla 3: Tabla comparativa de *Clúster Computing*, *Grid Computing* y *Cloud Computing*.

Después de examinar los tres tipos de topologías, la tecnología más atractiva para implementar en una *Smart City* es la plataforma de *cloud computing*. Una de las cualidades más importantes por las que este tipo de sistema distribuido parece óptimo para una ciudad inteligente es la capacidad de ofrecer directamente al ciudadano un conjunto muy amplio de servicios virtuales, sin que el ciudadano se percate de toda la



infraestructura, gestión de datos y administración que hay detrás.

También permite la adhesión de nuevos servicios de manera sencilla y de manera escalable; los servicios son proporcionados por servidores en red encargados de atender las peticiones de los usuarios en cualquier momento. Los ciudadanos y las empresas, en definitiva, los usuarios pueden tener acceso a su información o servicio mediante una conexión a internet desde cualquier dispositivo móvil o fijo ubicado en cualquier lugar.

En *cloud computing* se une la potencia de muchos servidores independientes en una red de servidores. Las peticiones que llegan, no solamente van a un servidor, si no que se distribuyen al conjunto de servidores con esto se produce lo que se llama balanceo de carga, es decir se reparte el volumen de procesamiento necesario para mejorar el rendimiento.

Si un nodo del *cloud* falla es invisible para el usuario ya que las máquinas virtuales pueden replicarse fácilmente y mantener sin problemas la estabilidad del sistema, esta redundancia de información permite no perder datos en casos de actualizaciones de datos o restauraciones de máquinas.

En definitiva, concluyo que un sistema distribuido gestionado por una topología de *cloud computing* es la mejor opción para implementar una *Smart City*, que podría ofrecer tanto a la ciudad como a los propios ciudadanos los siguientes servicios<sup>16</sup>, entre otros:

- Servicios de transporte y movilidad para reducir la congestión del tráfico, ahorrar tiempo y desperdicio de combustible y reducir la contaminación, tanto atmosférica como acústica.
- Energía y servicios ambientales que controlan las emisiones de gases de efecto invernadero.
- Servicios de seguridad pública.
- Coordinación y programación de obras públicas, y las reparaciones

---

<sup>16</sup> <https://www.bu.edu/hic/research/scope/>

municipales.

- Herramientas para la gestión de los activos de la ciudad a través de la minería de grandes conjuntos de datos y la coordinación del uso activo de multitud de fuentes.
- Servicios sociales, institucionales y mecanismos de comportamiento para facilitar la adopción de nuevos servicios, tales como programas y campañas de incentivos y reportes de la comunidad para promover la transparencia y la sostenibilidad.

### **3.2.1. Alternativas de Cloud Computing**

Mediante *cloud computing* los ciudadanos de la *Smart City* pueden obtener servicios e infraestructura de manera virtual, rápida, y transparente a toda la tecnología que da el soporte, tal y como se ha mencionado anteriormente. Para generar un entorno de *cloud* en *Smart City* necesitamos un *software* potente, que permita escalabilidad y que ofrezca a los usuarios los recursos necesarios para la creación de servicios; la tecnología necesaria para este propósito debe estar basada en *Infrastructure as a Services*.

El objetivo será poder administrar a los usuarios, en este caso a empresas, una infraestructura completa virtual en la que únicamente deban proponer e implementar un servicio al que tendrán acceso los ciudadanos.

Para aprovechar el máximo rendimiento se necesita un *software* capaz de ofrecer tecnología red, virtualización de infraestructura, almacenamiento de datos, replicación de servicios en caso de pérdida y gestión mediante un panel web virtual entre otros.

Un punto importante y destacable es utilizar tecnología *open-source*, hay que pensar que no se trata únicamente de ser una tecnología gratuita, sino que permite que el *software* sea modificado según las necesidades del proveedor de servicios; al ser libre esta soportado por toda la comunidad y permite evolucionar rápidamente, es decir estar actualizado y adaptándose a nuevas tecnologías y a las nuevas necesidades.



Figura 10<sup>17</sup>: Alternativas de *Cloud Computing*: *OpenStack*, *Eucalyptus*, *OpenNebula* y *CloudStack*.

Existen diversos *softwares* libres<sup>18</sup> basados en tecnología *IaaS* que comparten las características mencionadas anteriormente. Entre las más destacadas cabe mencionar: *OpenNebula*, *CloudStack* y *Eucalyptus* (Ver Figura 10), sin embargo, el más demandado y apoyado por la comunidad es *OpenStack*, que está en constante desarrollo.

Además su popularidad es agigantada, el proyecto inicial empezó en el año 2010 por la empresa Rackspace Cloud y por la agencia espacial norteamericana, NASA.

Actualmente, alrededor de 200 empresas se han unido al proyecto, entre las que se encuentran empresas tan importantes como *AMD*, *Intel*, *Canonical*, *SUSE Linux*, *Red Hat*, *IBM*, *Dell*, *HP*, *Cisco*, etc. En septiembre de 2012, cuando se unieron gran parte de las empresas más importantes del mercado de las *TIC* se dio lugar al proyecto *OpenStack* dirigido por la *OpenStack Foundation*, como se conoce ahora, de código abierto y que da la posibilidad a empresas y usuarios, de crear nuevas plataformas libres en la nube y con infinidad de posibilidades.

En los últimos dos años de funcionamiento la Fundación ya ha atraído a más de 9.500

<sup>17</sup> <http://evaluandocloud.com/estudio-comparativos-de-plataformas-cloud-computing/>

<sup>18</sup> [http://posgrado.frba.utn.edu.ar/investigacion/especialidades/Bocchio-2013\\_tf\\_esp.pdf](http://posgrado.frba.utn.edu.ar/investigacion/especialidades/Bocchio-2013_tf_esp.pdf)

miembros individuales procedentes de 100 países y 850 organizaciones diferentes. *OpenStack* parece ser la opción óptima para conseguir un sistema distribuido, que ofrezca servicios a los usuarios para construir una tecnología capaz de gestionar una *Smart City*.

Las ventajas de *OpenStack*<sup>19</sup> que pueden mencionarse son las de toda propuesta *open-source* en general, como el ahorro de costes, la mayor independencia frente a soluciones propietarias, una mayor modularidad o mayores opciones de personalización, pero también tiene algunas desventajas, como la dificultad de instalar e implementar.

El tipo de industria a la que se dirige es muy variado y provienen de disciplinas muy distantes aparentemente, como pueden ser las telecomunicaciones, *SaaS* y *e-commerce* hasta finanzas o salud. La idea es que estas empresas sean más ágiles, reduzcan costes y eviten dependencias de proveedores.

Además, todo el código de *OpenStack* está disponible libremente bajo la licencia *Apache 2.0*, por ello, presenta las ventajas propias del *software* libre, haciendo que cualquier persona puede ejecutar, construir sobre ella, o presentar cambios al proyecto.

### 3.2.2. Introducción a *OpenStack*

*OpenStack*<sup>20</sup> es una plataforma *software cloud computing* de libre distribución para desplegar nubes públicas y privadas, desarrollada con la idea de ser sencilla de gestionar, masivamente escalable y con muchas prestaciones. *OpenStack*, como ya se ha dicho anteriormente proporciona una solución de Infraestructura como servicio (*IaaS*) a través de un conjunto de servicios interrelacionados.

*OpenStack* controla grandes volúmenes de información, almacenamiento y recursos de red a través de un centro de datos, todo ello gestionado a través de un panel de control que proporciona a los administradores capacidad de controlar los recursos que se ofrecen a los usuarios.

---

<sup>19</sup> <https://openwebinars.net/que-es-eso-de-openstack-por-que-deberia-conocerlo/>

<sup>20</sup> <https://es.wikipedia.org/wiki/OpenStack>

*OpenStack*<sup>21</sup> pertenece a la fundación *OpenStack Foundation* que ha optado por utilizar la licencia de *software* libre, para aprovechar al máximo las contribuciones de la comunidad sobre el proyecto. La fundación debe tomar las decisiones finales de desarrollo.

Todo el código del proyecto *OpenStack* está disponible a través de *github*, además de *múltiples* aportaciones propias de distintos usuarios; la metodología de desarrollo de *OpenStack* también merece la pena que se destaque, ya que gracias a la utilización de metodologías ágiles se consigue que cerca de 1000 personas repartidas por todo el mundo hayan participado en el desarrollo de las distintas versiones.

Todo el código de *OpenStack* está desarrollado en *Python*, lenguaje de programación completamente libre y con una enorme progresión en los últimos años.

Podemos destacar los principios fundamentales sobre los que se basa *OpenStack Foundation*:

- Licencia *Apache 2.0*
- Proceso de diseño abierto
- Repositorios públicos de código fuente
- Todos los procesos de desarrollo deben estar documentados y ser transparentes
- Orientado para adoptar estándares abiertos
- Diseño modular que permite flexibilidad mediante el uso de *APIs*

## ESTRUCTURA MODULAR

*OpenStack* no es un solo producto, sino un conjunto de módulos que pueden combinarse en función de las características y necesidades de cada caso, hay algunos componentes fundamentales y otros opcionales y alternativos. Este esquema funcional es más potente que las estructuras privativas que incluyen un conjunto más o menos fijo de componentes muy bien integrados entre sí pero poco adaptables.

*OpenStack* tiene una gran flexibilidad y permite implementar un *cloud* privado en una pequeña empresa o algunos de los mayores *clouds* públicos del mundo, como contrapartida, exige que la persona encargada de la implementación y la

---

<sup>21</sup> <http://docs.openstack.org/>

administración del *cloud* tenga suficientes conocimientos en todas o la mayor parte de tecnologías subyacentes.

Cada componente de *OpenStack* es totalmente autónomo y funcional. Utiliza el protocolo *AMQP* de gestión de colas para comunicarse con el resto de componentes y una *API web RESTfull* para comunicarse con procesos "externos" o los usuarios.

Los principales componentes de *OpenStack* son los siguientes:

**HORIZON:** Proporciona un portal de autoservicio basado en web para interactuar con los servicios de *OpenStack* subyacentes, como el lanzamiento de una instancia, la asignación de direcciones *IP* y la configuración de los controles de acceso.

**NOVA:** Gestiona el ciclo de vida de instancias de un proceso. Las responsabilidades incluyen la creación, la programación y la destrucción de las máquinas virtuales dependiendo de la demanda.

Básicamente gestiona todas las actividades necesarias para apoyar el ciclo de vida de las instancias dentro de *OpenStack*. Esto hace que *Nova* sea una plataforma de gestión que administra los recursos de cómputo, redes, autorización, y las necesidades de escalabilidad.

Sin embargo, *Nova* no proporciona ninguna capacidad de virtualización por sí mismo, sino que utiliza las *API* de *libvirt* para interactuar con los *hypervisores* compatibles como *KVM*, *Xen* y *Hyper-V*.

Funciones y características de *NOVA*:

- Gestión del ciclo de vida de Instancias
- Gestión de recursos informáticos
- Permisología.
- *API* basado en *REST*
- Comunicación consistente, eventualmente asincrónica

**NEUTRON:** Es un sistema para la gestión de redes y direcciones *IP*. Asegura que la red no presente el problema del cuello de botella o el factor limitante en el despliegue del *cloud* y ofrece a los usuarios un autoservicio real, incluso a través de sus

configuraciones de red. Permite ofrecer *IPs* estáticas o *DHCP* reservados a petición del usuario.

El módulo de *networking* permite generar redes planas o virtuales *VLAN* para diferentes aplicaciones o grupos de usuarios para la separación de los servidores y el tráfico.

Los usuarios también pueden crear sus propias redes, controlar el tráfico y conectar los servidores y los dispositivos a una o más redes.

*OpenStack Networking* tiene un marco que permite la extensión de servicios de red adicionales, como los sistemas de detección de intrusos (*IDS*), balanceo de carga, cortafuegos y redes privadas virtuales (*VPN*) para ser implementada y administrada.

**KEYSTONE:** El servicio de Identidad de *OpenStack (Keystone)* ofrece un directorio central de usuarios asignados a los servicios de *OpenStack* que tienen permiso de acceso. Actúa como un sistema de autenticación común en todo el sistema operativo para la nube y se puede integrar con los servicios de directorio *backend* existentes como *LDAP*. Es compatible con múltiples formas de autenticación, incluyendo nombre de usuario y contraseña de credenciales estándar y sistemas basados en *tokens*.

**GLANCE:** *Glance* es el servicio de imágenes virtuales de *OpenStack*, proporciona servicios de búsqueda, almacenamiento y entrega de las imágenes para las máquinas virtuales. Las imágenes almacenadas se pueden utilizar como una plantilla. También se puede utilizar para almacenar y catalogar un número ilimitado de copias de seguridad. El servicio de imagen puede almacenar imágenes de disco y de servidores. La *API* de servicios de imagen proporciona una interfaz *REST* estándar para consultar información sobre las imágenes de disco y permite a los clientes transmitir las imágenes a nuevos servidores.

**RABBITMQ:** No es propiamente un módulo, pero es el método de comunicación utilizado por *OpenStack*, que se comunica entre sí utilizando la cola de mensajes. *Nova* utiliza llamadas asincrónicas para la solicitud de respuesta, con una devolución de llamada que se desencadena una vez que se recibe una respuesta. Dado que se utiliza la comunicación asincrónica, ninguna de las acciones del usuario se bloquea por mucho tiempo en un estado de espera. Esto es efectivo ya que muchas de las acciones previstas por la *API* de llamadas, tales como el lanzamiento de una instancia

o añadir una imagen, consume mucho tiempo.

Funciones y características de *RABBITMQ*:

- La mensajería es asíncrona, desacoplando las aplicaciones mediante la separación del envío y recepción de datos.
- Entrega de datos segura.
- Operaciones no bloqueantes.

Estos son los servicios básicos necesarios para la implementación y el funcionamiento de *OpenStack*. En la siguiente tabla (ver Tabla 4) se muestran un resumen básico de los servicios de los módulos mencionados hasta ahora.

MODULO	SERVICIO
<i>HORIZON</i>	Panel web de gestión
<i>NOVA</i>	Gestión de instancias
<i>NEUTRON</i>	Redes virtuales
<i>KEYSTONE</i>	Autenticación y autorización
<i>GLANCE</i>	Gestión de imágenes para las instancias
<i>RABBITMQ</i>	Gestión colas de mensajes.

Tabla 4: Tabla que indica los principales servicios de *OpenStack*.

Existen también los siguientes elementos que puede incorporarse al *cloud* ofreciendo así otros servicios adicionales.

**SWIFT:** *Swift* proporciona un almacén de objetos virtuales eventualmente consistentes distribuido para *OpenStack*. *Swift* es capaz de almacenar miles de millones de objetos distribuidos en diferentes nodos. *Swift* ha incorporado redundancia y tolerancia a fallos de gestión y es capaz de transmitir y guardar multimedia. Es sumamente escalable tanto en términos de tamaño lógico y de capacidad.

Funciones y características de *SWIFT*:

- Almacenamiento de gran cantidad de objetos
- Almacenamiento de objetos de tamaño grande
- Redundancia de datos
- Capacidad de Archivo



- Trabajo con conjuntos de datos grandes
- Contenedor de datos de máquinas virtuales y aplicaciones
- Capacidad de transmisión de multimedia
- Almacenamiento seguro de los objetos
- Copia de seguridad y archivado
- Escalabilidad extrema

**CINDER:** *OpenStack Block Storage (Cinder)* proporciona el *software* para crear y gestionar de forma centralizada un servicio de almacenamiento en forma de dispositivos de bloques conocidos como volúmenes *cinder*. En el escenario más común, los volúmenes *cinder* proporcionan un almacenamiento persistente para máquinas virtuales que son administrados por el *software OpenStack Compute*. *Cinder* también se puede utilizar de forma independiente de otros servicios de *OpenStack*.

*Cinder* permite a las organizaciones hacer un catálogo de dispositivos de almacenamiento basados en bloques con diferentes características. Por ejemplo, un tipo de potencial volumen de almacenamiento podría ser el almacenamiento de base de datos. *Cinder* también cuenta con capacidades de almacenamiento básicos como *snapshots* de máquinas.

*Cinder* fue originalmente un componente del proyecto *Nova*, conocido como "*nova-volume*".

**CEILOMETER:** El servicio de Telemetría de *OpenStack (Ceilometer)* proporciona un único punto de contacto para la facturación, la evaluación comparativa, la escalabilidad y con fines estadísticos, todo fácilmente auditable desde el *Dashboard*.

**SAHARA:** Proporciona capacidades para ofrecer y *Apache Hadoop* y *clústers* en *OpenStack* mediante la especificación de parámetros, la topología del clúster y datos del *hardware* de los nodos.

Tabla resumen de otros servicios (ver Tabla 5) de los módulos opcionales de *OpenStack*:

MODULO	SERVICIO
SWIFT	Almacenamiento de objetos virtuales
CINDER	Almacenamiento de máquinas virtuales
CEILOMETER	Control de estadísticas
SAHARA	Clustering

Tabla 5: Tabla que indica los servicios opcionales de *OpenStack*.

### Esquema Conceptual de OpenStack<sup>22</sup>

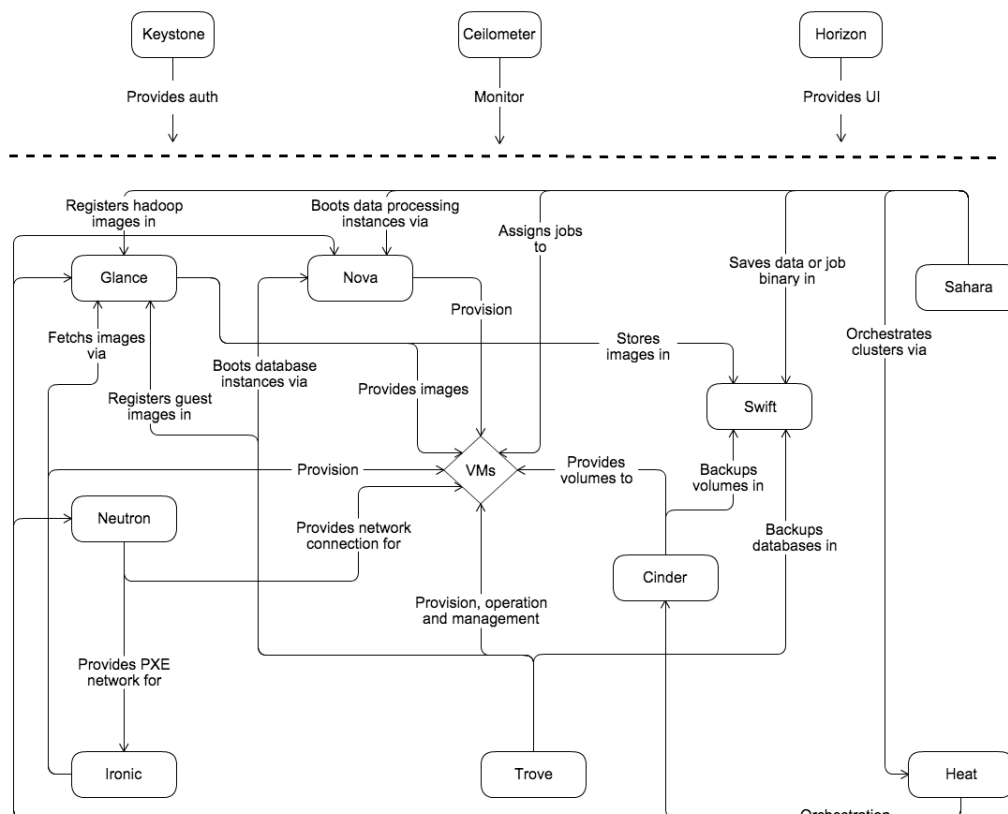


Figura 11: Esquema conceptual de la distribución de los módulos OpenStack.

<sup>22</sup> <http://docs.openstack.org/admin-guide/common/get-started-conceptual-architecture.html>

Para dar soporte al esquema conceptual (ver Figura 11) vamos a explicar cómo lanzar una instancia, para generar una máquina virtual. Hay que tener en cuenta que estos pasos son genéricos, ya que dependiendo de los componentes de cada *cloud* y de si estos están en el mismo o diferentes equipos, el proceso puede variar. Debemos pensar que estos pasos son realizados por el usuario, pero que el *backend* lo gestiona todo y es transparente, es decir el usuario utilizará una interfaz web amigable proporcionada por *Horizon*, nombrada *Dashboard* para facilitarle todo el proceso.

En primer lugar, el usuario se autentica mediante *Keystone*, o bien directamente a través del panel web *Horizon*. De esta manera obtiene un *token* de sesión que le permitirá realizar acciones con el resto de componentes de *OpenStack* sin necesidad de volver a autenticarse, estas acciones estarán limitadas por los permisos del rol que tenga asignado el usuario en *Keystone*, hay que recordar que cada servicio tiene su propia base de datos, aunque esta sea compartida.

El usuario solicita a *Glance* la lista de imágenes disponibles. Estas imágenes las tendrá alojadas *Glance* directamente en el equipo que se ejecuta o bien serán objetos almacenados en *Swift*.

El usuario selecciona una imagen y solicita a *Nova* que la instancie, para lo que *Nova* le pedirá las características de la instancia (ram, disco, *cpu*, etc.) y una vez recibidas las características, elegirá el nodo del *cloud* más adecuado para ejecutar la instancia.

De la configuración de la red virtual donde se encuentra situada la instancia se encarga *Neutron* permitiendo no sólo que se pueda acceder a la instancia desde fuera, a través de la denominada *IP* flotante, sino incluso de la configuración de redes complejas.

Todos estos pasos se realizan rápidamente y en unos segundos el usuario tendrá a su disposición una instancia de la imagen seleccionada.

### **3.2.3. Segunda implementación práctica: *OpenStack* sobre *Raspberry Pi 2***

En este segundo test experimental del proyecto nos hemos propuesto el objetivo de instalar *OpenStack* en *Raspberry Pi 2*, y ver sus capacidades para gestionar tal

cantidad de carga, en un pequeño sistema distribuido de 2 máquinas.

En la documentación de *OpenStack* recomiendan que la máquina máster sobre la cual se instale el *software* debe disponer de 2 GB de *RAM*, aun así, hemos optado por intentar realizar el proceso de instalación e intentar resolver los problemas encontrados.

En un principio hemos accedido al repositorio *GitHub*, plataforma web muy concurrida por la comunidad para la realización y almacenamiento con control de versiones *git* de proyectos realmente interesantes y útiles, hemos empezado por la vertiente de buscar en esta plataforma de desarrollo colaborativo información para la instalación automática de los distintos módulos que engloba el programa *OpenStack*.

Hemos probado distintos instaladores de la comunidad sobre *Raspbian*, pero el resultado ha sido negativo, en todos los casos la respuesta era que la plataforma *OpenStack* no daba soporte al sistema operativo *Raspbian*.

En esta captura de pantalla (ver Figura 12) se puede observar la incompatibilidad de uno de los recursos de *GitHub*<sup>23</sup> utilizados:

```
pi@raspberrypi:~/OpenSTK/devstack $ ./stack.sh
WARNING: this script has not been tested on Raspbian-8.0.
[Call Trace]
./stack.sh:181:die
[ERROR] ./stack.sh:181 If you wish to run this script anyway run with FORCE=yes
/home/pi/OpenSTK/devstack/functions-common: line 128: /opt/stack/logs/error.log: No such file or directory
pi@raspberrypi:~/OpenSTK/devstack $ FORCE=yes ./stack.sh
WARNING: this script has not been tested on Raspbian-8.0.
[Call Trace]
./stack.sh:211:install_package
/home/pi/OpenSTK/devstack/functions-common:1080:real_install_package
/home/pi/OpenSTK/devstack/functions-common:1072:exit_distro_not_supported
/home/pi/OpenSTK/devstack/functions-common:159:die
[ERROR] /home/pi/OpenSTK/devstack/functions-common:159 Support for Raspbian-8.0. is incomplete: no support for installing packages
/home/pi/OpenSTK/devstack/functions-common: line 128: /opt/stack/logs/error.log: No such file or directory
pi@raspberrypi:~/OpenSTK/devstack $
```

Figura 12: Captura de pantalla donde se muestra la incompatibilidad con *Raspbian*.

<sup>23</sup> <https://github.com/OpenStack-dev/devstack/>  
<http://www.trescca.eu/index.php/2013-05-23-13-18-38/guides/118-raspberry-pi-as-compute-node-in-openstack.html>

En este punto el planteamiento ha cambiado, hemos decidido utilizar como sistema operativo *Ubuntu 14.04.5 LTS (Trusty Tahr)*<sup>24</sup>, que si soporta según la página oficial la versión *Liberty OpenStack*, la misma web<sup>25</sup> ofrece los pasos necesarios para la instalación con un paquete que instala todos los módulos.

Seguimos esos pasos, pero no se ha podido instalar debido a problemas entre las dependencias de paquetes (ver Figura 13) y la arquitectura *ARM* de *Raspberry Pi 2*.

```
root@ubuntu-controller:/home/ubuntu# apt-get install python-openstackclient
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 python-openstackclient : Depends: python-cinderclient (>= 1:1.3.1) but it is not going to be installed
                        Depends: python-glanceclient (>= 1:0.18.0) but it is not going to be installed
                        Depends: python-keystoneclient (>= 1:1.6.0) but it is not going to be installed
                        Depends: python-neutronclient (>= 1:2.6.0) but it is not going to be installed
                        Depends: python-novaclient (>= 2:2.28.1) but it is not going to be installed
                        Depends: python-oslo.serialization (>= 1.2.0) but it is not going to be installed
                        Depends: python-oslo.utils (>= 2.0.0) but it is not going to be installed
E: Unable to correct problems, you have held broken packages.
root@ubuntu-controller:/home/ubuntu#
```

Figura 13: Captura de pantalla donde se muestra errores con las dependencias.

Durante los varios intentos sin éxito para resolver los problemas encontrados, apareció una nueva versión del sistema operativo concretamente *Ubuntu 16.04.1 LTS (Xenial Xerus)*, así que decidimos utilizarla e instalar en ella *OpenStack*, las dificultades seguían surgiendo, esta vez el problema estaba en que la versión *Liberty* de *OpenStack* utilizada no era compatible con la versión del sistema operativo (ver Figura 14).

```
root@ubuntu-controller:/home/ubuntu# add-apt-repository cloud-archive:liberty
Ubuntu Cloud Archive for OpenStack Liberty
More info: https://wiki.ubuntu.com/ServerTeam/CloudArchive
Press [ENTER] to continue or ctrl-c to cancel adding it

cloud-archive for Liberty only supported on trusty
```

Figura 14: Captura de pantalla donde se muestra la incompatibilidad entre *Liberty* y *Ubuntu Trusty*.

Como se puede apreciar el mensaje en concreto es “*cloud-archive for liberty only supported on trusty*”, pero como mencionamos antes ya se testeó *OpenStack Liberty*

<sup>24</sup> <https://wiki.ubuntu.com/ARM/RaspberryPi>

<http://docs.openstack.org/juno/install-guide/install/apt/content/>

<sup>25</sup> <http://docs.openstack.org/juno/install-guide/install/apt/content/>

sobre *Ubuntu Trusty* y no dio resultado.

Visto la dificultad de instalar todos los paquetes del aplicativo con un instalador automático, procedimos a cambiar a un método más lento, pero más eficaz a mi parecer; reinstalamos el sistema operativo Ubuntu 16.04, instalamos y configuramos solamente los módulos necesarios de *OpenStack* para conseguir la comunicación entre las máquinas y la posibilidad de crear una máquina virtual compartida. Esta vez nos guiamos mediante la documentación oficial de la página web de *OpenStack*.<sup>26</sup>

*OpenStack* utiliza una base de datos para almacenar toda la información relacionada con la configuración de cada módulo, en la documentación específica que se puede utilizar el software de *MariaDB* o *MySQL* para la gestión de la base de datos, pero para la tecnología ARM no existen paquetes de *MariaDB*, así que utilizamos *MySQL*. Los módulos que hemos instalado han sido *RabbitMQ*, *Keystone*, *Glance*, *Nova*, *Glance* y *Dashboard*, por este orden. El proceso de instalación ha sido lento y difícil, ya que han ido apareciendo problemas que se han ido resolviendo. Lo que dificultó conseguir que *OpenStack* funcionara por completo fue que este *software* ocupa muchos más recursos de los que la *Raspberry Pi 2* dispone. El consumo de *CPU* (ver Figura 16) por los módulos de *nova* y *keystone* era altísimo, sin dejar atrás el consumo de la memoria *RAM* (ver Figura 15).

```
root@ubuntu-controller:/home/ubuntu# free -m
              total        used         free       shared    buff/cache   available
Mem:           925          768           67           6           89          126
Swap:           0             0             0
root@ubuntu-controller:/home/ubuntu#
```

Figura 15: Captura de pantalla donde se muestra el consumo de la memoria *RAM* por parte de los módulos de *OpenStack*.

<sup>26</sup> <http://docs.OpenStack.org/liberty/install-guide-ubuntu/>

```
Tasks: 130 total, 12 running, 118 sleeping, 0 stopped, 0 zombie
%Cpu(s): 95.0 us, 4.6 sy, 0.0 ni, 0.0 id, 0.2 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem : 948016 total, 14160 free, 752944 used, 180912 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 163768 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4938	nova	20	0	62784	53164	7728	R	97.9	5.6	0:23.83	nova-conduct+
5055	keystone	20	0	12880	11016	4320	R	93.6	1.2	0:02.76	keystone-all
4866	nova	20	0	79700	69688	8060	R	86.4	7.4	0:32.61	nova-schedul+
5043	nova	20	0	101064	80852	3260	R	20.0	8.5	0:01.56	nova-api
5039	nova	20	0	100808	80552	3260	R	12.9	8.5	0:01.25	nova-api
5040	nova	20	0	100808	80492	3260	R	12.9	8.5	0:01.23	nova-api
5037	nova	20	0	100808	80608	3260	R	12.1	8.5	0:01.31	nova-api
5038	nova	20	0	100808	80548	3260	R	12.1	8.5	0:01.26	nova-api
5045	nova	20	0	100552	80400	3260	R	12.1	8.5	0:01.10	nova-api
5036	nova	20	0	100808	80660	3260	R	11.4	8.5	0:01.36	nova-api
5044	nova	20	0	100552	80400	3260	R	11.4	8.5	0:01.08	nova-api
4705	nova	20	0	99528	86280	10312	S	4.3	9.1	0:38.90	nova-api
878	glance	20	0	78600	71608	7952	S	3.6	7.6	1:27.88	glance-api
399	mysql	20	0	1143924	50232	13220	S	2.9	5.3	0:04.88	mysqld
55	root	20	0	0	0	0	S	2.1	0.0	0:03.12	mmcqd/0
725	rabbitmq	20	0	130248	29108	2792	S	2.1	3.1	0:39.83	beam.smp
5063	root	20	0	5120	2360	1848	R	2.1	0.2	0:00.10	top
7	root	20	0	0	0	0	S	0.7	0.0	0:03.74	rcu_sched
19	root	20	0	0	0	0	S	0.7	0.0	0:00.23	ksoftirqd/3
358	root	20	0	3412	1488	1320	S	0.7	0.2	0:00.77	irqbalance
983	ubuntu	20	0	11908	2900	2160	S	0.7	0.3	0:02.22	sshd
3354	nova	20	0	74308	60964	9768	S	0.7	6.4	0:13.82	nova-novncpr+
1	root	20	0	5476	3792	2660	S	0.0	0.4	0:41.34	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.34	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.86	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.05	migration/1
11	root	20	0	0	0	0	S	0.0	0.0	0:00.26	ksoftirqd/1
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.85	migration/2
15	root	20	0	0	0	0	S	0.0	0.0	0:00.27	ksoftirqd/2
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/2:0H
18	root	rt	0	0	0	0	S	0.0	0.0	0:00.85	migration/3
21	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/3:0H

```
root@ubuntu-controller:/home/ubuntu#
```

Figura 16: Captura de pantalla donde se muestra el consumo de la CPU por parte de los módulos de *OpenStack*.

Todo ello dificultaba trabajar con la aplicación por la lentitud para cargar tanto el *Dashboard* como para ejecutar cualquier comando de consulta mediante la consola, así que, como *Raspberry Pi 2* no dispone de *swap*<sup>27</sup> pensamos en añadirle un par de GB para compartir la carga de memoria con la RAM, aun así, la mejoría fue ínfima, no se pudo resolver de otra manera. (ver Figuras 17 y 18)

<sup>27</sup> <https://kirbian.wordpress.com/2012/10/30/manual-ajusta-el-tamano-de-la-particion-de-debian-rpi/>  
<http://trastetes.blogspot.com.es/2014/06/como-activar-particion-swap-en-ubuntu.html>

```

ubuntu@ubuntu-controller: ~
Tasks: 119 total, 4 running, 115 sleeping, 0 stopped, 0 zombie
%Cpu(s): 60.3 us, 5.2 sy, 0.0 ni, 32.4 id, 2.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem : 948016 total, 257444 free, 522768 used, 167804 buff/cache
KiB Swap: 2048 total, 1321 free, 727 used. 395452 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 1341 nova      20   0   60180 50448 7620 R   94.4   5.3   0:09.84 nova-api
 1363 keystone  20   0   28636 25572 5364 R   94.4   2.7   0:04.95 keystone-all
 1079 nova      20   0   81780 69044 5440 S   33.3   7.3   0:03.02 nova-conduct+
 1082 nova      20   0   81780 69044 5440 R   33.3   7.3   0:03.02 nova-conduct+
 725 rabbitmq  20   0 137940 28916 3000 S   11.1   3.1   0:12.04 beam.smp
 1373 ubuntu    20   0   5016 2328 1856 R   11.1   0.2   0:00.06 top
 884 nova      20   0   79612 69716 8128 S    5.6   7.4   0:46.33 nova-conduct+
 1081 nova      20   0   81780 69044 5440 S    5.6   7.3   0:02.94 nova-conduct+
 1 root      20   0   5492 3824 2656 S    0.0   0.4   0:07.31 systemd
 2 root      20   0   0 0 0 S    0.0   0.0   0:00.00 kthreadd
 3 root      20   0   0 0 0 S    0.0   0.0   0:00.08 ksoftirqd/0
 4 root      20   0   0 0 0 S    0.0   0.0   0:00.00 kworker/0:0
 5 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 kworker/0:0H
 6 root      20   0   0 0 0 S    0.0   0.0   0:00.04 kworker/u8:0
 7 root      20   0   0 0 0 S    0.0   0.0   0:00.96 rcu_sched
 8 root      20   0   0 0 0 S    0.0   0.0   0:00.00 rcu_bh
 9 root      rt   0   0 0 0 S    0.0   0.0   0:00.82 migration/0
 10 root     rt   0   0 0 0 S    0.0   0.0   0:00.02 migration/1
 11 root     20   0   0 0 0 S    0.0   0.0   0:00.08 ksoftirqd/1
 12 root     20   0   0 0 0 S    0.0   0.0   0:00.00 kworker/1:0
 13 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 kworker/1:0H
 14 root     rt   0   0 0 0 S    0.0   0.0   0:00.81 migration/2
 15 root     20   0   0 0 0 S    0.0   0.0   0:00.08 ksoftirqd/2
 16 root     20   0   0 0 0 S    0.0   0.0   0:00.00 kworker/2:0
 17 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 kworker/2:0H
 18 root     rt   0   0 0 0 S    0.0   0.0   0:00.81 migration/3
 19 root     20   0   0 0 0 S    0.0   0.0   0:00.07 ksoftirqd/3
 20 root     20   0   0 0 0 S    0.0   0.0   0:00.00 kworker/3:0
 21 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 kworker/3:0H
 22 root     0 -20   0 0 0 S    0.0   0.0   0:00.00 khelper
 23 root     20   0   0 0 0 S    0.0   0.0   0:00.00 kdevtmpfs
 24 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 netns
 25 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 perf
 26 root     20   0   0 0 0 S    0.0   0.0   0:00.00 khungtaskd
 27 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 writeback
 28 root      0 -20   0 0 0 S    0.0   0.0   0:00.00 crypto
ubuntu@ubuntu-controller:~$
    
```

Figura 17: Captura de pantalla donde se muestra el consumo de la memoria RAM por parte de los módulos de *OpenStack* después de ampliar la swap.

```

root@ubuntu-controller:/home/ubuntu# free -m
              total          used         free       shared  buff/cache   available
Mem:           925            768           67           6           89          126
Swap:          2048             727          1321
root@ubuntu-controller:/home/ubuntu#
    
```

Figura 18: Captura de pantalla donde se muestra el consumo de la memoria RAM por parte de los módulos de *OpenStack* después de ampliar la swap.



Así que llegados a este punto se logró instalar *OpenStack* pero debido a los recursos consumidos, se hizo imposible crear una máquina virtual compartida.

En el Anexo B se ha construido un manual donde se puede encontrar todo el proceso de instalación y configuración realizado, en algunos puntos de la instalación han aparecido problemas que hemos ido resolviendo.

### 3.2.4. Resultados

En este test experimental, como ya hemos mencionado, los resultados no fueron los esperados, se trabajó para lograr un sistema distribuido compuesto por dos máquinas mediante *OpenStack*, pero una vez conseguido, el consumo de recursos ha imposibilitado la creación de una máquina virtual compartida. En las siguientes capturas podemos ver algunos de los detalles del *software* instalado.

En esta primera captura se pueden ver las interfaces tanto del *nodo máster (ubuntu-controller)* en el que se instalaron los módulos mencionados anteriormente, y el otro nodo (*ubuntu-node1*) en el que se instaló el módulo *Nova* que gestiona las máquinas virtuales.

También podemos ver mediante la consola del *controller* en el campo *Status* como el servicio principal del módulo de *Nova (nova-compute)* se comporta al detenerlo y encenderlo desde el otro nodo.

## Creación de un clúster/cloud a partir de Raspberry Pi para un sistema distribuido

Xavier Hernández Villahermosa

```
root@ubuntu-node1: /home/ubuntu
root@ubuntu-node1: /home/ubuntu#
root@ubuntu-node1: /home/ubuntu# service nova-compute start
root@ubuntu-node1: /home/ubuntu# service nova-compute status
nova-compute.service - OpenStack Compute
Loaded: loaded (/lib/systemd/system/nova-compute.service; enabled; vendor prese
Active: active (running) since Tue 2016-05-31 20:54:52 CEST; 3s ago
Process: 1428 ExecStartPre=/bin/chown nova:nova /var/lock/nova /var/log/nova /va
Process: 1425 ExecStartPre=/bin/mkdir -p /var/lock/nova /var/log/nova /var/lib/n
Main PID: 1431 (nova-compute)
CGroup: /system.slice/nova-compute.service
        aa1431 /usr/bin/python /usr/bin/nova-compute --config-file=/etc/nova/no
May 31 20:54:52 ubuntu-node1 systemd[1]: Starting OpenStack Compute...
May 31 20:54:52 ubuntu-node1 systemd[1]: Started OpenStack Compute.
root@ubuntu-node1: /home/ubuntu#
root@ubuntu-node1: /home/ubuntu#
root@ubuntu-node1: /home/ubuntu# service nova-compute stop
root@ubuntu-node1: /home/ubuntu# service nova-compute status
nova-compute.service - OpenStack Compute

root@ubuntu-controller: /home/ubuntu
root@ubuntu-controller: /home/ubuntu#
root@ubuntu-controller: /home/ubuntu# nova-manage service list
DEPRECATED: Use the nova service-* commands from python-novaclient instead or the os-services REST resource. The service
Binary      Host              Zone      Status      State Updated_At
nova-scheduler ubuntu-controller internal enabled      (-) 2016-05-31 18:55:50
nova-osapi_compute 0.0.0.0          internal enabled      XXXX None
nova-metadata 0.0.0.0          internal enabled      XXXX None
nova-cert     ubuntu-controller internal enabled      (-) 2016-05-31 18:55:51
nova-consoleauth ubuntu-controller internal enabled      (-) 2016-05-31 18:55:51
nova-conductor ubuntu-controller internal enabled      (-) 2016-05-31 18:55:51
nova-compute  ubuntu-node1     nova      enabled      (-) 2016-05-31 18:55:53
root@ubuntu-controller: /home/ubuntu# nova-manage service list
DEPRECATED: Use the nova service-* commands from python-novaclient instead or the os-services REST resource. The service
Binary      Host              Zone      Status      State Updated_At
nova-scheduler ubuntu-controller internal enabled      (-) 2016-05-31 18:57:50
nova-osapi_compute 0.0.0.0          internal enabled      XXXX None
nova-metadata 0.0.0.0          internal enabled      XXXX None
nova-cert     ubuntu-controller internal enabled      (-) 2016-05-31 18:57:51
nova-consoleauth ubuntu-controller internal enabled      (-) 2016-05-31 18:57:51
nova-conductor ubuntu-controller internal enabled      (-) 2016-05-31 18:57:51
nova-compute  ubuntu-node1     nova      enabled      XXXX 2016-05-31 18:56:33
root@ubuntu-controller: /home/ubuntu# date
```

Figura 19: Captura de pantalla donde se muestra los servicios del módulo *Nova*.

En la siguiente captura se ejecuta el comando que permite crear máquinas virtuales por terminal, como ya hemos mencionado esto ha volcado un error, aunque el problema parezca que es producido por el módulo de *nova* no es así, ya que posteriormente se revisó todo empezando por los *logs* y estaba todo correctamente configurado.

```
root@ubuntu-controller:~# nova boot --image "cirros" --flavor ml.tiny myFirstVM
ERROR (ClientException): Unexpected API Error. Please report this at http://bugs.launchpad.net/nova/ and attach the Nova API log if possible.
<class 'oslo_db.exception.CantStartEngineError'> (HTTP 500) (Request-ID: req-ee5e86ae-eb69-4b6b-bd7e-4a3b9a6b8d26)
root@ubuntu-controller:~#
```

Figura 20: Captura de pantalla donde se muestra el error al intentar crear una máquina virtual en *OpenStack*.

## Creación de un clúster/cloud a partir de Raspberry Pi para un sistema distribuido Xavier Hernández Villahermosa

A continuación, se muestran algunas de las capturas realizadas de la interfaz *Dashboard* de *OpenStack* una vez instalado.

*Login* para la *API OpenStack*.

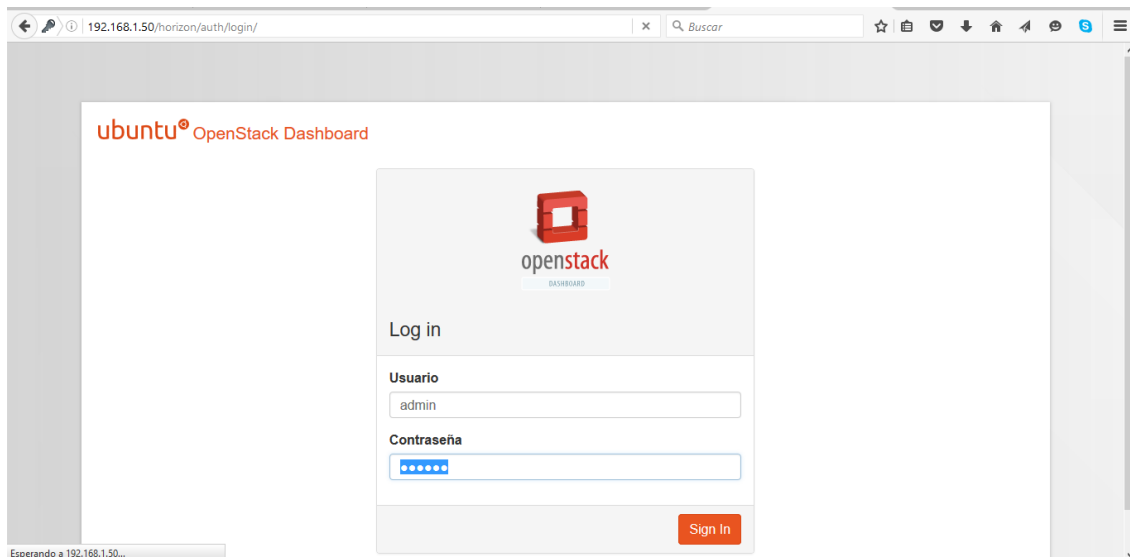


Figura 21: Login para la *API OpenStack*.

Servicio de gestión de los que se encarga, en este caso, el nodo “*controlador*”.

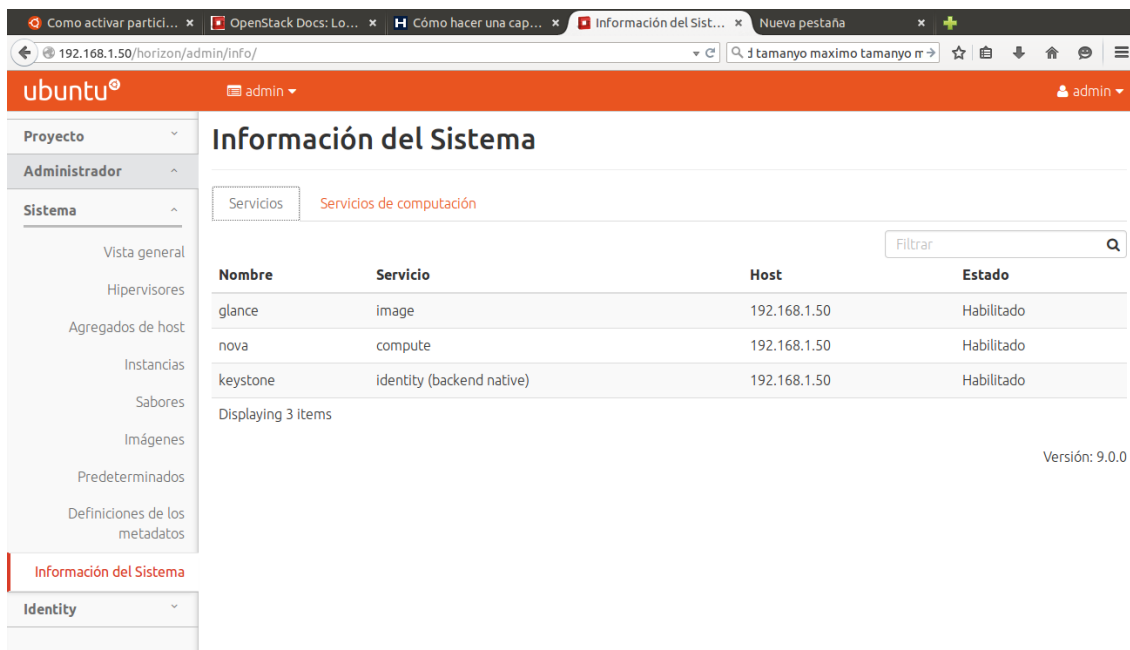
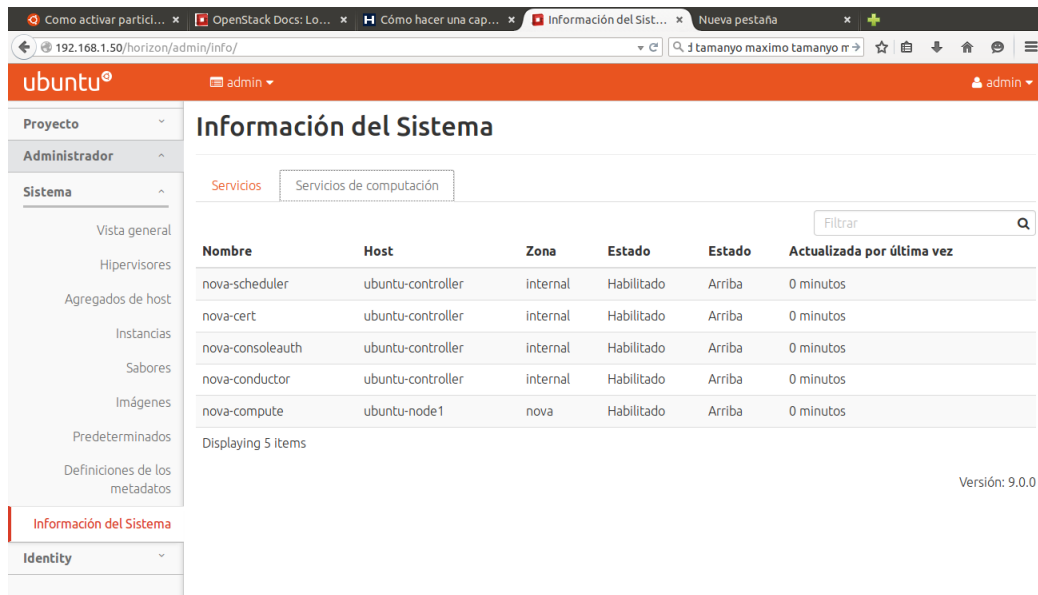


Figura 22: Servicios de gestión de la *API OpenStack*.

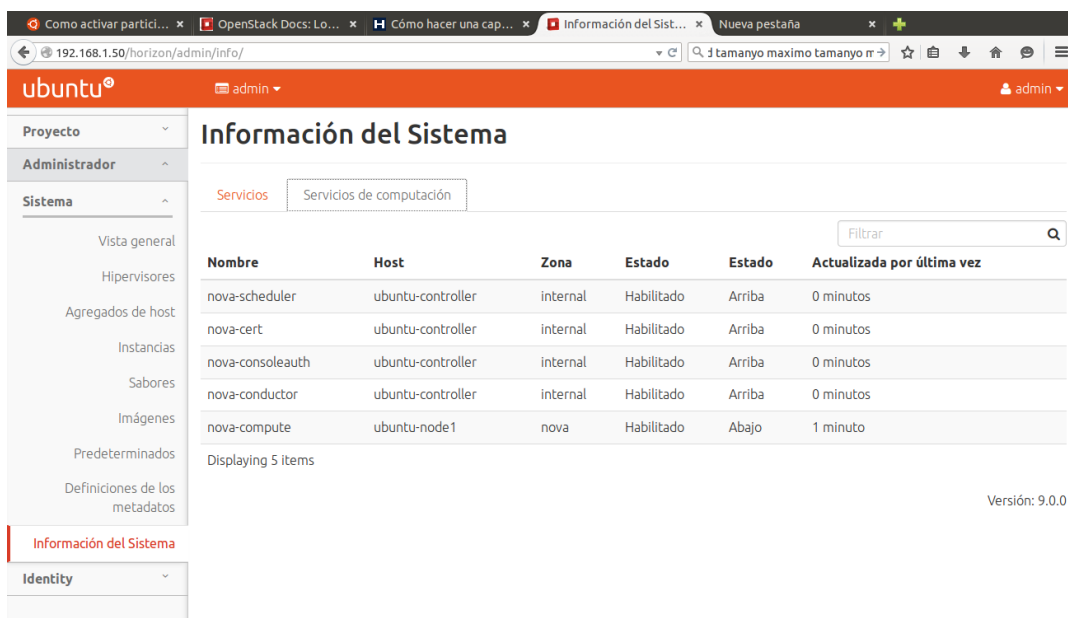
Las dos siguientes capturas parecen la misma, pero se tomaron en el momento de detener el servicio de *nova-compute*, así que en la primera vemos la tupla con los parámetros “Nombre -> nova-compute con Estado -> Arriba” y en la segunda vemos la misma tupla pero con el “Estado -> Abajo”.



The screenshot shows the OpenStack Horizon admin interface. The left sidebar contains navigation options: Proyecto, Administrador, Sistema, Vista general, Hipervisores, Agregados de host, Instancias, Sabores, Imágenes, Predeterminados, and Definiciones de los metadatos. The main content area is titled 'Información del Sistema' and shows a table of services under the 'Servicios de computación' tab. The table has columns for Nombre, Host, Zona, Estado, Estado, and Actualizada por última vez. The 'nova-compute' service is listed with Host 'ubuntu-node1', Zona 'nova', Estado 'Habilitado', and Estado 'Arriba'.

Nombre	Host	Zona	Estado	Estado	Actualizada por última vez
nova-scheduler	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-cert	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-consoleauth	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-conductor	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-compute	ubuntu-node1	nova	Habilitado	Arriba	0 minutos

Figura 23: Servicio *nova-compute* del módulo *Nova* encendido.



The screenshot shows the OpenStack Horizon admin interface, similar to the previous one, but the 'nova-compute' service status is now 'Abajo' (Down). The table shows the following data:

Nombre	Host	Zona	Estado	Estado	Actualizada por última vez
nova-scheduler	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-cert	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-consoleauth	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-conductor	ubuntu-controller	internal	Habilitado	Arriba	0 minutos
nova-compute	ubuntu-node1	nova	Habilitado	Abajo	1 minuto

Figura 24: Servicio *nova-compute* del módulo *Nova* apagado.

En las siguientes capturas podemos ver 3 de los elementos esenciales para levantar una máquina virtual. Primero las imágenes, en este caso solo aparece la imagen que se quiso utilizar para el test realizado que se puede encontrar en el Anexo B. La imagen “*cirros*” es un sistema operativo que apenas pesa *13MB*, podrían administrarse imágenes de sistemas operativos más complejos con una implementación a gran escala.

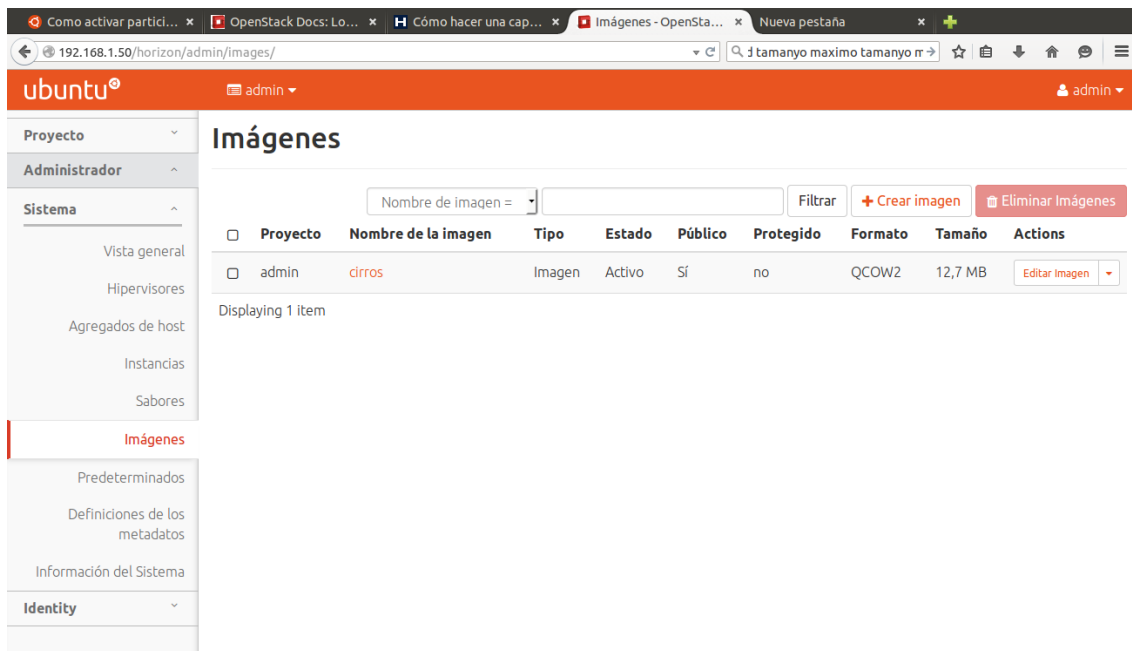


Figura 25: Servicio gestor de imágenes Glance.

Los sabores son los valores *hardware* virtualizados de los que podrá disponer la máquina virtual generada. Son valores customizable y se pueden crear de nuevos.

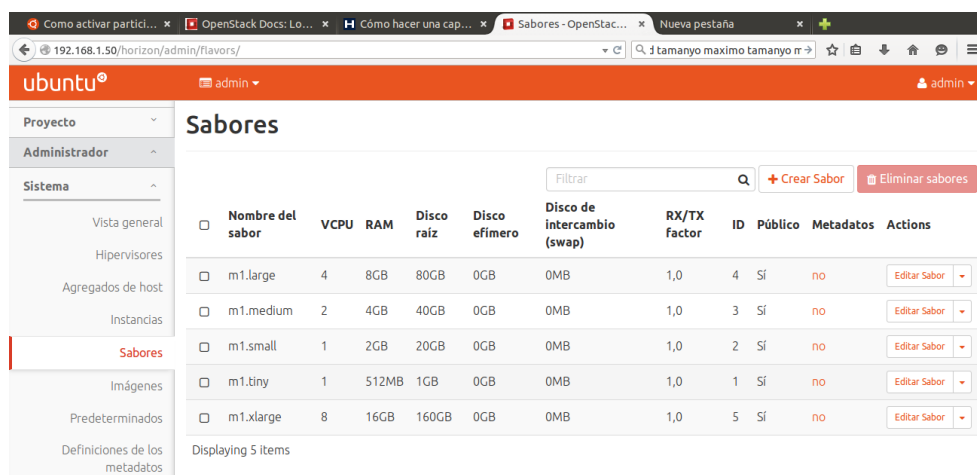


Figura 26: Infraestructura virtual para el aprovisionamiento de máquinas.

Por último, el tipo de emulador que permite virtualizar la máquina, y que nodo puede gestionarlo.

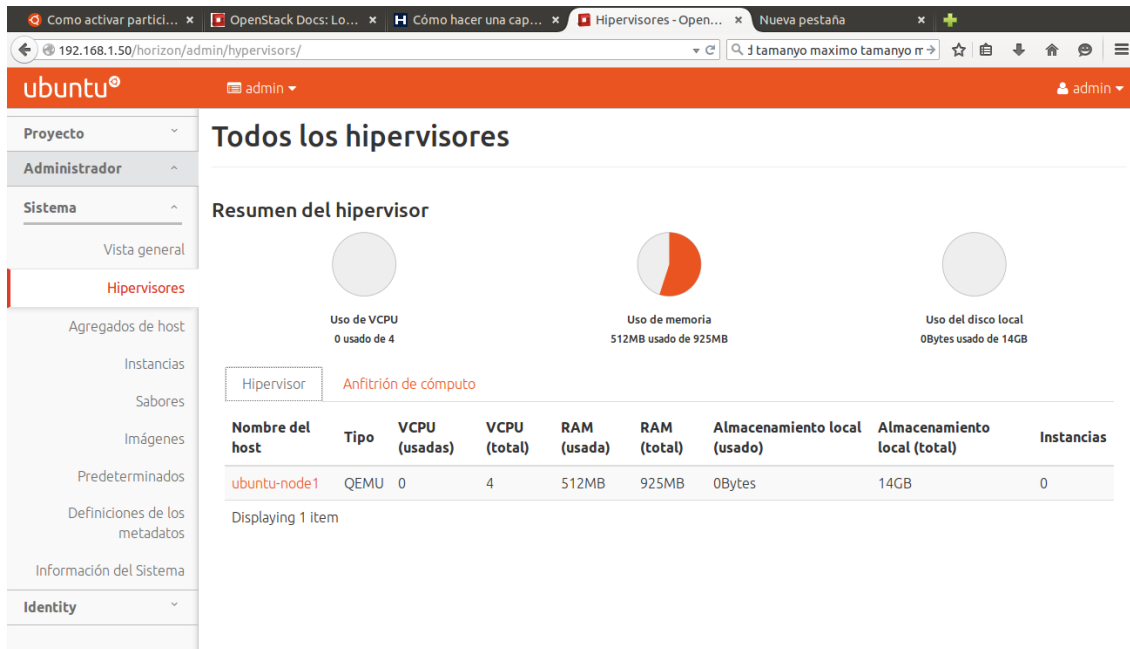


Figura 27: Captura de pantalla donde se muestra los nodos con *hipervisores*.

En esta última captura se pueden ver los errores que aparecían al intentar crear la máquina virtual desde el gestor del *Dashboard*. No eran errores estáticos, es decir en otras ocasiones el problema en vez de ser el siguiente: *“Error:No ha sido posible obtener los grupos de seguridad”* podía variar y decía que no ha sido posible encontrar una imagen o un sabor, cuando si existían en cada caso.

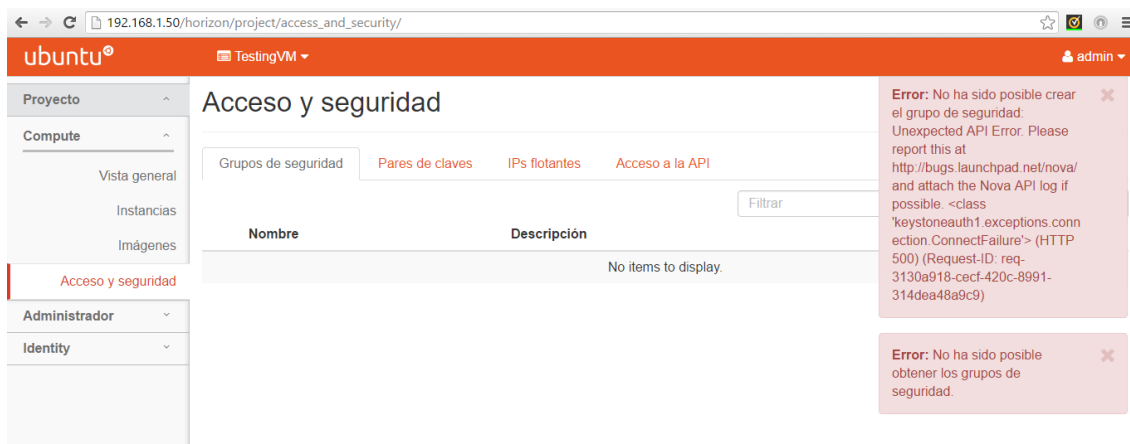


Figura 28: Captura de pantalla donde se muestra el error al crear la máquina virtual desde la *API*.

Como se menciona anteriormente el problema no se pudo resolver, así que no se llegaron a poder crear máquinas virtuales. Como resultado de este segundo test experimental sobre las capacidades de la *Raspberry* podemos concluir que para el caso del *software OpenStack*, las máquinas no pueden dar soporte a todos los recursos necesarios, aun así, se llegó a instalar OpenStack y a tener comunicación entre los distintos nodos. Para conseguir implementar una *Smart City con Cloud Computing* y con este tipo de placas se necesitaría utilizar otro software o placas más potentes.

### 3.3. Ejecución paralela en *clúster*

La computación *cloud* mediante el gestor de *OpenStack* no ha dado buenos resultados, como hemos podido comprobar los recursos necesarios para la implementación y ejecución del *software* han superado las capacidades de las *Raspberry Pi 2*, así que en este punto contemplamos la alternativa en cambiar de paradigma y apostar por la programación distribuida en topología *clúster*.

En apartados anteriores ya hemos definido la programación distribuida y las características de un *clúster*; en esta sección nos centraremos en definir que se interpreta con la ejecución paralela, que ventajas presenta y que modelo de sistema distribuido se podría llevar a cabo con este nuevo paradigma.

Aunque la tecnología ha avanzado mucho, todavía existen programas construidos a partir de un algoritmo que se ejecuta secuencialmente, donde cada instrucción es procesada por la *CPU* del ordenador, una vez termina, puede ejecutar la siguiente instrucción. La ejecución secuencial en muchos casos es necesaria y óptima, por ejemplo, para resolver problemas sencillos o programas que utilicen datos estáticos; la cosa se complica al trabajar con volúmenes grandes de información donde se necesita de una capacidad de procesamiento elevada tanto para evitar problemas en el sistema como para conseguir los resultados lo más rápido posible, para lograr esto hablamos de ejecución paralela.

La ejecución paralela permite que las instrucciones de un programa sean ejecutadas simultáneamente, para conseguir esto, se debe dividir el problema en partes más pequeñas de modo que cada sección pueda ser gestionada por elementos de procesamiento que ejecutan su parte del algoritmo de manera simultánea con los

otros, y finalmente agrupan los resultados. Cuando hablamos de elementos de procesamiento son aquellos componentes o recursos como ordenadores con múltiples procesadores (paralelizar por procesador), ordenadores en red (paralelizar entre *PCs*), paralelizar por *hardware* especializado, o simplemente se podrían combinar las distintas formas mencionadas. Estos elementos pueden utilizar la computación paralela en distintos puntos específicos: paralelismo a nivel de bit, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas.

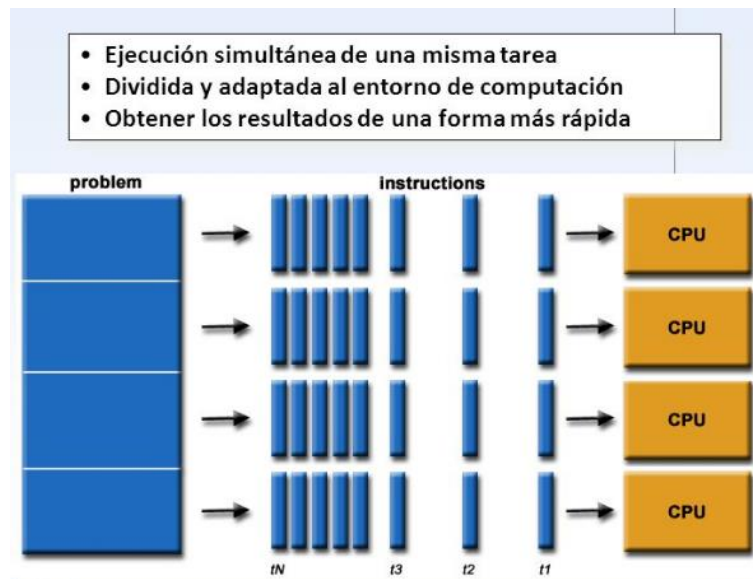


Figura 29: Captura de pantalla donde se muestra la división de un bloque en procesos.

Aunque hace años la computación paralela presentó algunas limitaciones como el consumo de energía y el aumento de temperatura en las máquinas, el cambio en la actualidad es enorme, se está trabajando para conseguir arquitecturas especializadas para la programación y ejecución en paralelo. Un ejemplo de ello son los estandarizados ordenadores con multiprocesador.

Las computadoras paralelas pueden clasificarse según el nivel de paralelismo que admite su *hardware*: equipos con procesadores multinúcleo y multiprocesador que tienen un conjunto de elementos de procesamiento dentro de una sola máquina y los *clústeres*, que utilizan varios equipos para trabajar en la misma tarea.

Nuestro objetivo a lograr ahora es un poco diferente al que se ha planteado al principio del proyecto. La capacidad necesaria para construir un *cloud* en que uno de los objetivos principales es ofrecer servicios al usuario (en caso de una *Smart City* a los ciudadanos), mediante miles máquinas virtuales que deberían ser autogestionadas por



el *software* automáticamente y de manera transparente y rápida para el usuario, hemos visto que es difícilmente tratable con la placa *Raspberry Pi 2*. Ahora el enfoque es distinto, como hemos mencionado anteriormente hay dos formas de entender quién es el cliente del sistema distribuido en una *Smart City*, pueden ser directamente los ciudadanos, o bien indirectamente, en el caso en que los datos recogidos y calculados son obtenidos por otras máquinas que toman decisiones para mejorar la vida del ciudadano.

Necesitamos un *software* que ofrezca mayor rendimiento, que utilice menos recursos para ser ejecutado, capaz de tratar con múltiples datos y distribuir las tareas de manera paralela para aprovechar el máximo de todas las máquinas que formaran parte del *clúster*.

Para ello hemos optado por la programación mediante un estándar de C llamado *MPI* (*Message Passing Interface*) que cumple las características mencionadas y que veremos a continuación.

### **3.3.1. Introducción *MPI* (*Message Passing Interface*)**

*MPI* (Interfaz de Paso de Mensajes)<sup>28</sup> es un lenguaje estándar que engloba un conjunto de bibliotecas para utilizar el paso de mensajes como protocolo de comunicación entre computadoras. Las rutinas desarrolladas dan soporte a los lenguajes de programación en *C*, *C++*, *Fortran* y *Ada*. La semántica y la sintaxis utilizadas hacen de *MPI* un estándar portátil, es decir, capaz de correr en casi todo tipo de arquitectura que permita trabajar con memoria distribuida sin perder rendimiento, es un protocolo rápido ya que fue diseñado para ser usado en programas que exploten la existencia de múltiples procesadores, además no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

Desde que se desarrolló el estándar final entre 1993 y 1994, el *MPI* ha ido creciendo y evolucionando en el mundo de la tecnología y se han implementado una gran variedad de plataformas que van desde las máquinas masivamente paralelas hasta redes de estaciones de trabajo, pasando por Supercomputadoras, como Caléndula gestionada por la Fundación del Centro de Supercomputación de Castilla y León, uno de los ordenadores más potentes de España desde 2009.

---

<sup>28</sup> [http://webdelprofesor.ula.ve/ingenieria/hhoeger/Introduccion\\_MPI.pdf](http://webdelprofesor.ula.ve/ingenieria/hhoeger/Introduccion_MPI.pdf)

El objetivo de *MPI* es desarrollar un estándar para ser ampliamente usado que permita escribir programas usando el paso de mensajes, técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua.



El estándar<sup>29</sup> es extenso en cuanto al número de rutinas que se especifican, contiene alrededor de 129 funciones muchas de las cuales tienen numerosas variantes y parámetros. Para aprovechar las funcionalidades extendidas de *MPI* se requieren muchas funciones (ver Tabla 6). Esto no implica una relación directamente proporcional entre la complejidad y el número de funciones, muchos programas paralelos pueden ser escritos usando sólo 4 funciones básicas. Sin embargo, *MPI* permite flexibilidad cuando sea requerida, además de que no se debe ser un experto en todos los aspectos de *MPI* para usarlo satisfactoriamente. No entraremos en detalle en explicar todas y cada una de las funciones del estándar, ya que no es el propósito del proyecto, pero si mencionaremos en forma de tabla las más utilizadas e importantes. En el siguiente apartado se explican las funciones de la biblioteca que se han usado para realizar el test final del proyecto.

<b>FUNCIONES IMPORTANTES DE MPI</b>	
<code>#include "mpi.h"</code>	Fichero <i>header file</i> necesario que provee los tipos, subrutinas y definiciones básicas de <i>MPI</i> .
<b>LLAMADAS UTILIZADAS PARA INICIALIZAR, ADMINISTRAR Y FINALIZAR COMUNICACIONES</b>	
<code>MPI_Init(&amp;argc, &amp;argv)</code>	Permite iniciar una sesión <i>MPI</i> . Esta función debe ser utilizada antes de llamar a cualquier otra función de <i>MPI</i> .
<code>MPI_Finalize()</code>	Permite terminar una sesión <i>MPI</i> . Esta función debe ser la última llamada a <i>MPI</i> . Permite liberar la memoria usada por

<sup>29</sup> [https://es.wikipedia.org/wiki/Interfaz\\_de\\_Paso\\_de\\_Mensajes](https://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes)

	<i>MPI.</i>
<i>MPI_Comm_size(MPI_Comm comm, int *size)</i>	Permite determinar el número total de procesos que pertenecen a un comunicador.
<i>MPI_Comm_rank(MPI_Comm comm, int *rank)</i>	Permite determinar el identificador ( <i>rank</i> ) del proceso actual.
<i>MPI_Get_processor_name(hostname,&amp;namelen)</i>	Retorna el nombre del procesador sobre el cual se está corriendo al momento de hacer la llamada.
<b>LLAMADAS UTILIZADAS PARA TRANSFERIR DATOS ENTRE UN PAR DE PROCESOS.</b>	
<i>MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)</i>	Realiza el envío de un mensaje de un proceso fuente a otro destino. Bloqueante.
<i>MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)</i>	Rutina de recibimiento de un mensaje desde un proceso. No Bloqueante.
<i>MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)</i>	Realiza el envío de un mensaje de un proceso fuente a otro destino. No Bloqueante.
<i>MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)</i>	Rutina de recibimiento de un mensaje desde un proceso. No Bloqueante.
<b>LLAMADAS PARA TRANSFERIR DATOS ENTRE VARIOS PROCESOS.</b>	
<i>MPI_Barrier(MPI_Comm comm)</i>	Detiene la ejecución de cada proceso que la llama hasta que todos los procesos incluidos en el grupo asociado a <i>comm</i> la hayan llamado.

<p><i>MPI_Bcast(</i>void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)</p>	<p>Permite a un proceso enviar una copia de sus datos a otros procesos dentro de un grupo definido por un comunicador.</p>
<p><i>MPI_Scatter(</i>void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)</p>	<p>Un proceso raíz divide un mensaje en partes iguales y los envía individualmente al resto de procesos y a sí mismo.</p>
<p><i>MPI_Gather(</i>void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)</p>	<p>Hace que cada proceso, incluyendo al <i>root</i>, envíe el contenido de su buffer de envío (<i>inbuf</i>) al proceso <i>root</i>. El proceso <i>root</i> recibe los mensajes y los almacena en orden (según el rango del que envía los datos) en el buffer de recepción (<i>outbuf</i>).</p>
<p><i>MPI_Reduce(</i>void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)</p>	<p>Combina los elementos provistos en el <i>buffer</i> de entrada (<i>inbuf</i>) de cada proceso en el grupo, usando la operación <i>op</i> (puede ser una operación predefinida o definida por el usuario) y retorna el valor combinado en el <i>buffer</i> de salida (<i>outbuf</i>) del proceso <i>root</i>.</p>
<p><b>LLAMADAS UTILIZADAS PARA CREAR TIPOS DE DATOS DEFINIDOS POR EL USUARIO.</b></p>	
<p><i>MPI_Type_structt(</i>int count, int *array_of_blocklength, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)</p>	<p>Permite la replicación de datos hacia bloques de diferentes tipos.</p>

<i>MPI_Pack(void *inbuf, int incnt, MPI_Datatype datatype, void *outbuf, int outsize, int *position, MPI_Comm comm)</i>	Empaqueta distintos tipos de datos en memoria continua
---	--

Tabla 6: Tabla que muestra las funciones básicas de la biblioteca *MPI*.

Como se ha mencionado no entraremos en detalle en cada uno de los parámetros de las distintas rutinas que hemos visto, pero si es importante entender el funcionamiento general del envío y recepción de mensajes en una comunicación entre procesos *MPI*. Además, han aparecido algunos términos comunes entre las distintas funciones como son los comunicadores (*MPI\_Comm*), funciones bloqueantes y no bloqueantes y la comunicación por *buffers*.

La comunicación se realiza mediante el paso de mensajes entre distintos procesos a través de espacios de memoria nombrados *buffer* donde se almacenan los datos de salida o se almacenan los datos de entrada.

El argumento *MPI\_Comm* compartido entre las distintas rutinas engloba una colección de procesos que se definen al ejecutar el programa, estos procesos pueden enviarse mensajes entre sí. Existe una variable genérica *MPI* para definir que la comunicación sea posible entre todos los procesos asociados, esa variable es *MPI\_COMM\_WORLD*.

La transferencia de datos entre dos procesos puede utilizar funciones bloqueantes y no bloqueantes. En el primer caso se realiza la llamada y se bloquea hasta que la operación de comunicación se complete. En el otro caso se inicia la operación de transferencia, pero su finalización debe ser realizada de forma explícita mediante llamadas como *MPI\_Test* y *MPI\_Wait*. *MPI\_Wait* es una llamada bloqueante y retorna cuando la operación de envío o recepción se completa. *MPI\_Test* permite verificar si la operación de envío o recepción ha finalizado, esta función primero chequea el estado de la operación de envío o recepción y luego retorna.

Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear *buffers*, en los que se almacenan los mensajes a espera que un proceso los reciba.

Generalmente empleando este sistema, el proceso que envía mensajes sólo se bloquea o para, cuando finaliza su ejecución, o si el *buffer* está lleno. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución.

En conclusión, del presente apartado se muestra a continuación una tabla (ver Tabla 7) con las principales características generales más destacables del estándar *MPI* (*Message Passing Interface*):

<b>CARACTERÍSTICAS GENERALES</b>	
Los comunicadores combinan procesamiento en contexto y por grupo para seguridad de los mensajes.	
Seguridad en la manipulación de aplicaciones.	
<b>Manejo de ambiente.</b>	Temporizadores y sincronizadores. <ul style="list-style-type: none"> <li>- Inicializar y finalizar.</li> <li>- Control de errores.</li> <li>- Interacción con el ambiente de ejecución.</li> </ul>
<b>Comunicación punto a punto:</b>	Heterogeneidad para <i>buffers</i> estructurados y tipos de datos derivados.  Varios modos de comunicación: <ul style="list-style-type: none"> <li>- Normal, con y sin bloqueo.</li> <li>- Síncrono.</li> <li>- Listo, para permitir acceso a protocolos rápidos.</li> <li>- Retardados, utilizando <i>buffers</i>.</li> <li>-</li> </ul>
<b>Comunicaciones colectivas:</b>	Capacidad de manipulación de operaciones colectivas con operaciones propias o definidas por el usuario.

	<p>Gran número de rutinas para el movimiento de datos.</p> <p>Los subgrupos pueden definirse directamente o por la topología.</p>
<b>Topologías por procesos:</b>	Soporte incluido para topologías virtuales para procesos (mallas y grafos).
<b>Caracterización de la Interface:</b>	Se permite al usuario interceptar llamadas <i>MPI</i> para instalar sus propias herramientas.

Tabla 7: Tabla que muestra las características generales del estándar *MPI*.

En la última implementación, instalaremos el estándar *MPI* para comprobar la capacidad de las *Raspberrys Pi 2* y conseguir la gestión de datos de manera distribuida para mejorar en rendimiento y velocidad. Tal implementación es un prototipo de *clúster* que podría evolucionar en un futuro a gran escala para crear un sistema distribuido en una *Smart City*. Ya no hablaríamos de un servicio directamente hacia el usuario sino de servicios automáticos que, dependiendo de los resultados obtenidos por los cálculos de los nodos, situados en los distintos puntos de la ciudad, se actuaría en consecuencia.

### **3.3.2. Tercera implementación práctica: *MPI* sobre *Raspberry Pi 2***

En este tercer y último test sobre *Raspberry Pi 2* se ha instalado y configurado el estándar *MPI* para la gestión de sistemas distribuidos, además se ha implementado un programa que multiplica matrices en lenguaje C, y el mismo programa se ha paralelizado mediante *MPI*.

La multiplicación de matrices (ver Figura 30) se realiza de la siguiente manera:

$$\begin{aligned} A \cdot B &= \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \\ &= \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 & 2 \cdot 0 + 0 \cdot 2 + 1 \cdot 1 & 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 & 3 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 & 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 & 5 \cdot 0 + 1 \cdot 2 + 1 \cdot 1 & 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{pmatrix} = \\ &= \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix} \end{aligned}$$

Figura 30: Figura que muestra una multiplicación de dos matrices cuadradas de 3x3.

Para la implementación y paralelización del programa se ha utilizado la siguiente lógica:

- División de la matriz A por filas
- Envío a cada máquina del clúster las filas de A que le corresponden según la división de procesos indicados en el momento de la ejecución
- Envío de toda la matriz B a todas las máquinas necesaria para realizar las operaciones.
- Cada máquina realiza el cálculo de una parte de los valores de la matriz final.
- Todos los resultados obtenidos por parte de cada máquina son enviados a la máquina en la que se ha ejecutado el programa, que los unifica y devuelve la matriz final.

Las funciones de la biblioteca *MPI* utilizadas han sido las siguientes:

***MPI\_Init(&argc, &argv)***: Función de inicialización del programa para *MPI*

***MPI\_Comm\_rank(MPI\_COMM\_WORLD, &idProc)***: Obtiene el número del



proceso(&idProc) que está en ejecución del conjunto total de procesos (MPI\_COMM\_WORLD).

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &numProc):** Obtiene el número total de procesos(&numProc) definidos del conjunto total de procesos (MPI\_COMM\_WORLD).

**MPI\_Get\_processor\_name(hostname,&namelen):** Obtiene el nombre de la máquina (hostname) en la que esta ejecutándose el proceso.

**MPI\_Bcast:** Con esta función se envía toda la matriz B a todos los dispositivos asociados al clúster, en este caso a 2 Raspberrys Pi 2.

**MPI\_Bcast(matrizB, SIZE\*SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Scatter:** Con esta función se divide la matriz A en filas según el número de procesos definidos y las filas correspondientes son enviadas individualmente al resto de máquinas y así misma.

**MPI\_Scatter (&matrizA[0][0], SIZE\*SIZE/numProc, MPI\_INT, MPI\_IN\_PLACE, SIZE\*SIZE/numProc, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Gather:** Recoge y unifica los datos enviados por todos los procesos que se han ejecutado tanto en la Raspberry en la que se ha ejecutado el programa como los procesos ejecutados en la otra máquina.

**MPI\_Gather (MPI\_IN\_PLACE, SIZE\*SIZE/numProc, MPI\_INT, &matrizR[0][0], SIZE\*SIZE/numProc, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Finalize():** Termina la ejecución del programa en MPI.

En este caso la instalación no fue tan complicada como en *OpenStack*, algo evidente ya que *MPI* es únicamente una biblioteca de funciones que combinadas con lenguajes como C permite crear programas paralelos. Para ello se utilizó parte de una guía de pasos que siguió la universidad de *Southampton*<sup>30</sup> para crear un supercomputador con

---

<sup>30</sup> [http://www.southampton.ac.uk/~sjc/raspberrypi/pi\\_supercomputer\\_southampton.htm](http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm)  
<http://www.xataka.com/makers/raspberry-pi-como-pequeno-supercomputador-bueno-bonito-y-barato>

*RBP.*

Aun así, hemos creado un manual propio con algunas modificaciones y adaptado a nuestras necesidades, que se puede encontrar en el Anexo C.

Además, para que un programa paralelizado con *MPI* se ejecute en distintas máquinas, estas deben tener también el ejecutable instalado, para ello hemos creado un script en *bash* para realizar la tarea de la copia del ejecutable de manera remota, esto permite que desde una única máquina que podría ser nuestro centro de mando o administración podamos crear o recibir y ejecutar los programas sin necesidad de acceder explícitamente a los demás nodos.

Como al principio del proyecto hemos detectado que el mejor sistema operativo para utilizar sobre *Raspberry Pi 2* es *Raspbian*, ha sido este el que hemos utilizado en esta ocasión para las máquinas.

Para realizar la instalación y configuración de *MPI* se han realizado básicamente tres pasos.

1. La instalación del *software* descargado desde la web oficial<sup>31</sup> de *MPICH*.
2. Configuración de las variables de entorno para definir el directorio el que encontrar los binarios.
3. Creación de un fichero para definir los miembros que forman parte del sistema distribuido.

En este punto ya tenemos funcionando *MPI* en un nodo, para añadir otro simplemente debemos de copiar una imagen del sistema operativo del nodo funcional, cambiar la IP y crear un par de claves *ssh* para conseguir una relación de confianza.

En el Anexo C se puede encontrar el manual que hemos generado para instalar *MPI* paso a paso entre dos máquinas, y los programas y script correspondientes.

---

<sup>31</sup> <http://www.mpich.org>

### 3.3.3. Resultados

Este proceso experimental ha dado resultados verdaderamente positivos, se ha instalado perfectamente la librería *MPI* en las máquinas, y se ha podido comprobar cómo se ha dividido la carga de trabajo al ejecutar un programa que se ha elaborado para multiplicar matrices de pequeñas y grandes dimensiones, mediante *MPI*. Para facilitar la distribución de programas en el clúster de *Raspberrys* se ha generado un fichero *bash* básico. *MPI* permite paralelizar de distintos modos, se puede paralelizar un programa en una sola máquina ejecutándolo y definiendo el número de procesos en que quieres que se divida la ejecución, en ese caso y dependiendo de la implementación el sistema paralelizará el código, y por lo tanto irá más rápido. Puede darse el mismo caso pero que tengamos más de una máquina en el sistema como en el clúster de dos máquinas que hemos implementado, en ese caso no solo paralelizará el código en distintos procesos, sino que además lo distribuirá entre las diferentes máquinas, en ese caso la velocidad de cálculo será más rápida y el programa terminará antes.

Para comprobar todo esto, a continuación, se muestra una tabla con los resultados obtenidos para distintos casos de los tiempos de cálculo para las multiplicaciones de matrices cuadradas, de 100x100, 300x300, 500x500 y de 700x700, esta última calculando 343000000 multiplicaciones y 342510000 sumas, un total de 685510000 operaciones.

Valores en segundos.	Secuencial	Paralelo 1 proc 1 máq	Paralelo 2 proc 2 máq's	Paralelo 20 proc 1 máq	Paralelo 20 proc 2 máq's
<b>M (100x100)</b>	0.038533	0.037215	0.018103	0.002014	0.001920
<b>M (300x300)</b>	1.091648	1.067866	0.513496	0.300297	0.153165
<b>M (500x500)</b>	6.365902	6.119386	3.084469	1.549765	0.743546
<b>M (700x700)</b>	18.772628	18.02920	9.001999	4.649466	2.209201
<b>1000x1000</b>	<i>Segmentation fault</i>	62.73557	31.30697	16.22502	8.227989

Tabla 8: Tabla que muestra los tiempos de cálculo para la multiplicación de matrices en *MPI* con distintos escenarios.

Si observamos la tabla (ver Tabla 8) se aprecia como el tiempo disminuye, cada vez que el programa es segmentado, ya sea a nivel de procesos, como en la ejecución conjunta con otra máquina del clúster.

## 4. CONCLUSIONES

Los objetivos principales del proyecto se han realizado con éxito, pero los resultados finales no han sido los esperados. Se ha sometido a la placa *Raspberry Pi 2* a distintas pruebas experimentales entre ellas la capacidad para virtualizar, con los resultados obtenidos se puede afirmar que, aunque se ha conseguido emular la arquitectura x86 ejecutando Windows 98, la *Single Board Computer* no tiene la capacidad ni los recursos para soportar toda gestión del sistema operativo inquilino y ofrecer un sistema, funcional y fluido.

Otra prueba ha sido instalar el *software OpenStack* para la gestión de un sistema distribuido para *Cloud Computing*, en este caso y a pesar de la dificultades en el proceso de instalación y de los obstáculos encontrados se pudo terminar instalando el *software* mencionado, aun así la capacidad de cálculo y rendimiento necesarios para hacer de *OpenStack* un herramienta funcional no han sido soportados por la *Raspberry Pi 2*, es decir no se puede gestionar los recursos que *Openstack* puede ofrecer, como las máquinas virtuales.

Al ver la dificultad para emular y virtualizar en *cloud*, se planteó realizar otra prueba, dicha prueba ha consistido en crear un sistema distribuido con el estándar de *MPI* que da soporte a distintos lenguajes de programación como *C*. La instalación ha sido sencilla y fluida, y ha permitido crear un *clúster* entre dos *Raspberrys*. Posteriormente se ha implementado un programa en lenguaje *C* que puede multiplicar matrices densas. Al ejecutarlo en el *clúster* se ha comprobado que ha distribuido perfectamente el trabajo de cómputo, y ha respondido con resultados correctos y mucho más rápido que si solo operase en una sola placa.

Con los resultados obtenidos, y a día de hoy puede ser complicado e ineficaz construir un sistema distribuido *Cloud* para cubrir las necesidades de una *Smart City* mediante las *SBC Raspberry Pi 2*

Como alternativa, se podría implementar infraestructura en red compuesta de *routers* para la comunicación entre nodos *Raspberry Pi 2*, y los nodos conectados a distintos sensores obteniendo datos del entorno. Las placas realizarían los cálculos pertinentes

y los resultados se mandarían a un *Data Center*. Posteriormente desde otros servidores se podría utilizar esos datos para dar servicios al ciudadano.

Podemos decir que para llegar a conseguir crear un sistema distribuido con todas las características que ofrece *cloud computing* mediante *Raspberry Pi 2* u otra *Single Board Computer* del mercado actual, habrá que esperar a que las placas evolucionen un poco más.

## 5. AGRADECIMIENTOS

Este proyecto experimental ha sido realizado de manera escalonada por la siguientes dos vertientes: la de plantear un sistema distribuido con *SBC (Raspberry Pi 2)* para una *Smart City*, y la parte práctica de testear las capacidades de la *Raspberry* para soportar distintas funcionalidades necesarias de un sistema distribuido, como son la virtualización, el soporte de un *software* potente de *cloud computing*, e implementar un sistema mediante el estándar *MPI* para distribuir la carga de datos.

En todo el proceso de elaboración del proyecto me gustaría hacer mención y agradecer la colaboración de Eva Marin tutora del proyecto, por todo el soporte teórico/técnico proporcionado y la supervisión del proyecto. Quiero agradecer también a Josep Farré, la posibilidad de trabajar en su despacho (Centro de Procesamiento de Datos) *CPD* en el edificio Neapolis de Vilanova y la Geltrú, donde permitió que se conectara el *clúster* de *Raspberry Pi 2* con *OpenStack*, y se configuró para acceder remotamente. Ambos también han proporcionado la mayor parte del material y las herramientas utilizadas para realizar los test prácticos.

Se han citado varias reuniones, con Josep Farré y Eva Marin y siempre han estado dispuestos a dar críticas constructivas que han ayudado a ir madurando el proyecto.

## 6. BIBLIOGRAFIA

### Soporte para *Hardware*

- 17.09.2016 [https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB\\_comp.jpeg](https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_comp.jpeg)
- 30.02.2016 [https://es.wikipedia.org/wiki/Arquitectura\\_ARM](https://es.wikipedia.org/wiki/Arquitectura_ARM)
- 28.04.2016 [https://ca.wikipedia.org/wiki/Advanced\\_RISC\\_Machines](https://ca.wikipedia.org/wiki/Advanced_RISC_Machines)
- 05.05.2016 <http://www.xatakahome.com/trucos-y-bricolaje-smart/probamos-la-nueva-raspberry-pi-2-a-fondo>
- 15.06.2016 <http://es.slideshare.net/williamflazh/definiciones-sobre-switch-router-modem-wimax-acces-point>
- 17.09.2016 <http://i1.wp.com/guzman6001.byethost10.com/wp-content/uploads/2013/02/campos-del-encabezado-ipv4.png>
- 18.09.2016 [https://sites.google.com/site/redescena2cisco/\\_/rsrc/1342855927655/envio-depaquetes/Nueva%20imagen%20\(6\).bmp](https://sites.google.com/site/redescena2cisco/_/rsrc/1342855927655/envio-depaquetes/Nueva%20imagen%20(6).bmp)

### Soporte para *Software*

- 05.03.2016 <http://www.welivesecurity.com/la-es/2014/07/28/virtualizacion-o-emulacion-esa-es-la-cuestion/>
- 10.03.2016 <https://eltechs.com/product/exagear-desktop/>
- 12.03.2016 <http://hackerboards.com/emulator-brings-x86-linux-apps-to-arm-devices/>
- 23.03.2016 [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page)
- 30.03.2016 <http://mogaal.com/articulos/qemu.html>
- 02.04.2016 <https://launchpad.net/ubuntu/trusty/+package/qemu-system-x86>
- 27.09.2016 <http://gridcpm.blogspot.com.es/>
- 30.09.2016 <http://stackoverflow.com/questions/9723040/what-is-the-difference-between-cloud-grid-and-cluster>
- 02.10.2016 [http://www.garudaindia.in/html/pdf/ggoa\\_2011/day%201/cluster\\_grid\\_cloud\\_concepts.pdf](http://www.garudaindia.in/html/pdf/ggoa_2011/day%201/cluster_grid_cloud_concepts.pdf)
- 23.09.2016 <http://www.ijarcce.com/upload/2014/march/IJARCCE9B%20%20a%20%20anjan%20A%20Comparative%20Analysis%20Grid%20Cluster%20and%20Cloud%20Computing.pdf>
- 30.08.2016 <https://aws.amazon.com/es/types-of-cloud-computing/>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.465.8919&rep=rep1&type=pdf>

04.08.2016 <https://www.bu.edu/hic/research/scope/>

30.08.2016 <http://evaluandocloud.com/estudio-comparativos-de-plataformas-cloud-computing/>

22.09.2016 [http://posgrado.frba.utn.edu.ar/investigacion/especialidades/Bocchio-2013\\_tf\\_esp.pdf](http://posgrado.frba.utn.edu.ar/investigacion/especialidades/Bocchio-2013_tf_esp.pdf)

15.07.2016 <https://openwebinars.net/que-es-eso-de-openstack-por-que-deberia-conocerlo/>

10.03.2016 <https://es.wikipedia.org/wiki/OpenStack>

12:03.2016 <http://docs.openstack.org/>

30.03.2016 <http://docs.openstack.org/admin-guide/common/get-started-conceptual-architecture.html>

01.03.2016 <https://github.com/OpenStack-dev/devstack/>

05.03.2016 <http://www.trescca.eu/index.php/2013-05-23-13-18-38/guides/118-raspberry-pi-as-compute-node-in-openstack.html>

13.04.2016 <https://wiki.ubuntu.com/ARM/RaspberryPi>

07.04.2016 <http://docs.openstack.org/juno/install-guide/install/apt/content/>

20.04.2016 <http://docs.OpenStack.org/liberty/install-guide-ubuntu/>

18.07.2016 <https://kirbian.wordpress.com/2012/10/30/manual-ajusta-el-tamao-de-la-particion-de-debian-rpi/>

18.07.2016 <http://trastetes.blogspot.com.es/2014/06/como-activar-particion-swap-en-ubuntu.html>

20.08.2016 [http://webdelprofesor.ula.ve/ingenieria/hhoeger/Introduccion\\_MPI.pdf](http://webdelprofesor.ula.ve/ingenieria/hhoeger/Introduccion_MPI.pdf)

25.08.2016 [https://es.wikipedia.org/wiki/Interfaz\\_de\\_Paso\\_de\\_Mensajes](https://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes)

02.04.2016

[http://www.southampton.ac.uk/~sjc/raspberrypi/pi\\_supercomputer\\_southampton.htm](http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm)

02.04.2016 <http://www.xataka.com/makers/raspberry-pi-como-pequeno-supercomputador-bueno-bonito-y-barato>

## 7. ANEXO

### ANEXO A:

#### MANUAL NUMERO 1 EMULACIÓN

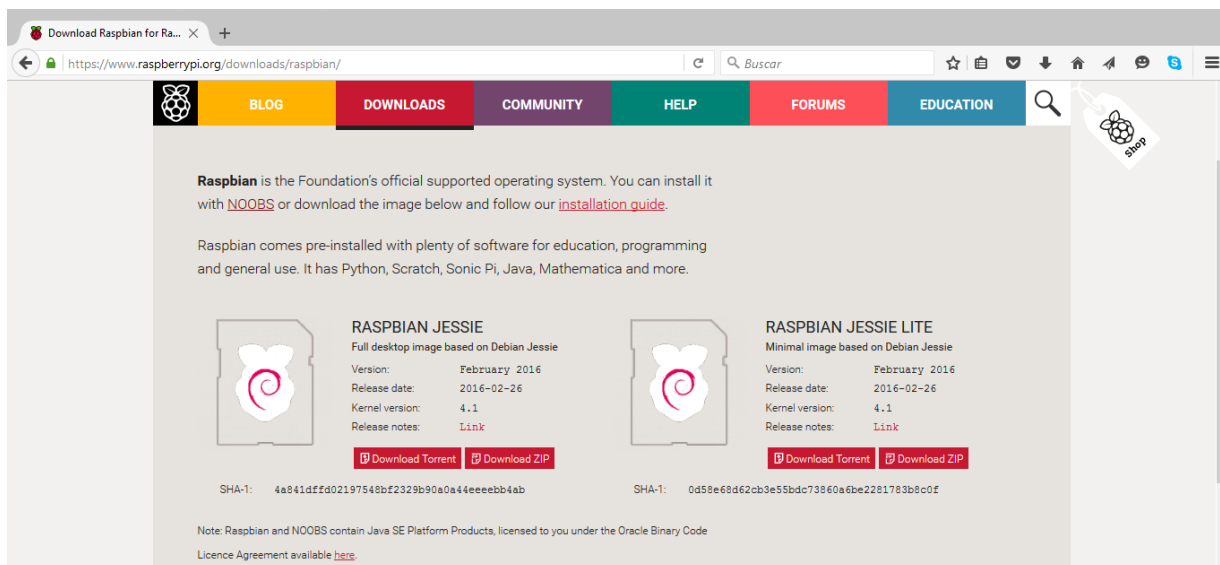
---

- INSTALACIÓN DE *RASPBIAN EN RASPBERRY*
- CONFIGURACIÓN DE *IP ESTÁTICA*
- EMULACIÓN DE SISTEMA OPERATIVO *x86 (WIN98)*

#### Instalación del Sistema Operativo:

---

1. Descargamos el fichero.zip del sistema operativo *Raspbian Jessie Lite* de la página oficial <https://www.raspberrypi.org/downloads/raspbian/>



Manual 1: Descarga del SO Raspbian

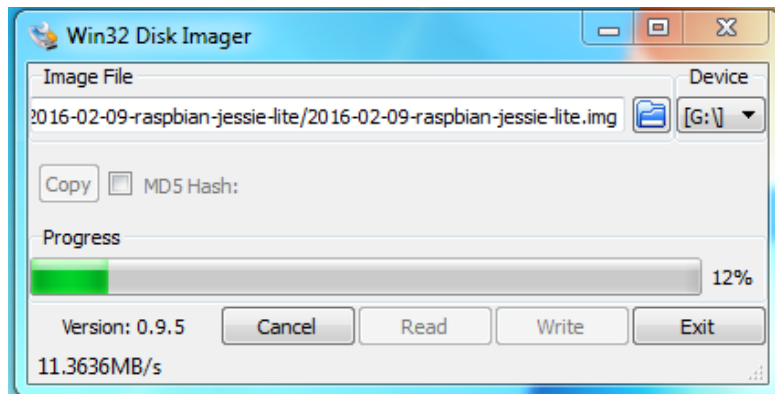
2. Descomprimos el **.zip**, y obtenemos la imagen (**2016-02-09-raspbian-jessie-lite.img**) que grabaremos en la **tarjeta SD**, para ello he utilizado el programa **Win32DiskImager** de *Windows*.



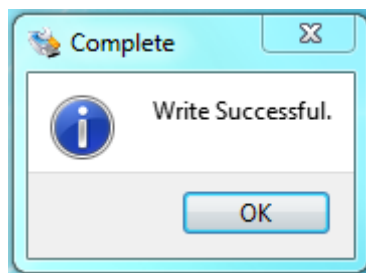
Podemos descargar el programa en el siguiente

enlace: <https://sourceforge.net/projects/win32diskimager/files/latest/download> .

Ejecutamos **Win32DiskImager** y elegimos la opción **Write**.



Manual 1: Grabar el SO en la tarjeta SD



Manual 1: Escritura completa sobre la SD

Podemos ver en **Equipo** que nuestra tarjeta SD de 16G ha reducido su tamaño, ahora vamos a configurar el teclado a Español, recuperar el espacio perdido y configurar algunos aspectos.

Ya tenemos la tarjeta *SD bootable* con el **SO Raspbian** instalado, solo queda insertar la tarjeta en la *Raspberry* y encenderla para empezar a trabajar.

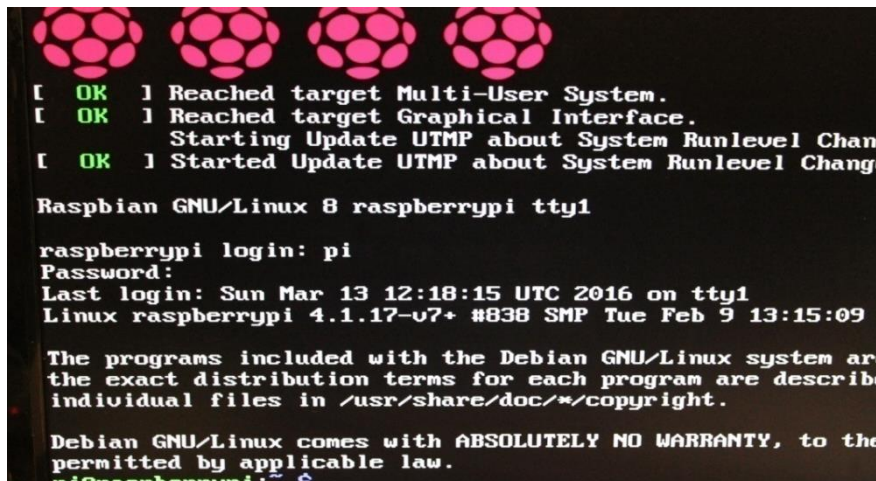
## Configuración básica del Sistema

---

Como hemos dicho el sistema ha arrancado y ahora nos pide un usuario y una contraseña, por defecto:

User: **pi**

Password: **raspberry**



```
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
Starting Update UTMP about System Runlevel Change
[ OK ] Started Update UTMP about System Runlevel Change

Raspbian GNU/Linux 8 raspberrypi tty1

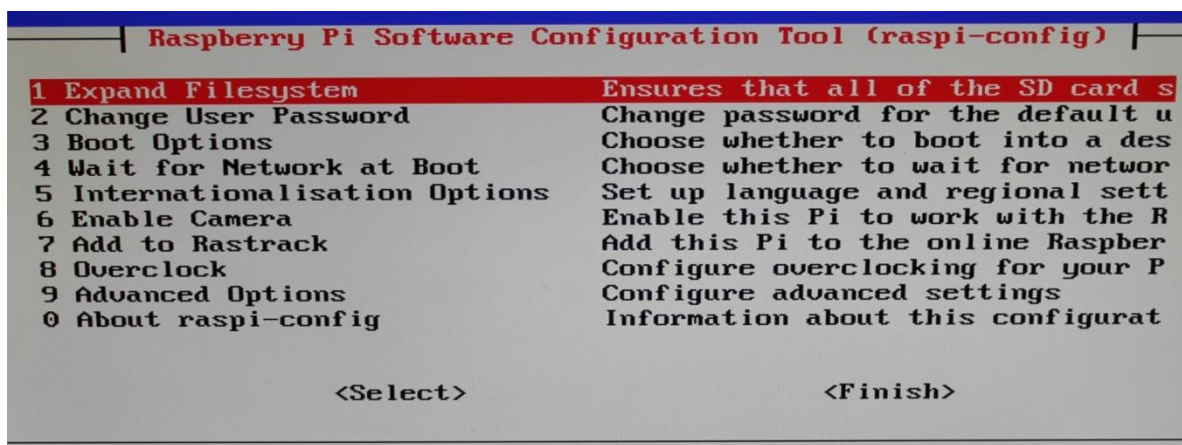
raspberrypi login: pi
Password:
Last login: Sun Mar 13 12:18:15 UTC 2016 on tty1
Linux raspberrypi 4.1.17-v7+ #838 SMP Tue Feb 9 13:15:09

The programs included with the Debian GNU/Linux system are
the exact distribution terms for each program are describe
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the
permitted by applicable law.
pi@raspberrypi:~$
```

Manual 1: Sistema iniciado

Una vez dentro del sistema vamos a entrar en el “modo” de configuración para *Raspbian* con el comando **raspi-config**. No hay que olvidarse de reiniciar la *Raspberry* (p.ej. **sudo shutdown -r now**) después de modificar su configuración para que los cambios se cometan.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Expand Filesystem          Ensures that all of the SD card s
2 Change User Password      Change password for the default u
3 Boot Options              Choose whether to boot into a des
4 Wait for Network at Boot  Choose whether to wait for networ
5 Internationalisation Options Set up language and regional sett
6 Enable Camera             Enable this Pi to work with the R
7 Add to Rastrack          Add this Pi to the online Raspber
8 Overclock                 Configure overclocking for your P
9 Advanced Options         Configure advanced settings
0 About raspi-config        Information about this configurat

<Select>                                <Finish>
```

Manual 1: Opciones de configuración SO

Podemos ver hasta 10 herramientas configurables, en mi caso únicamente

hablaremos de aquellas que vamos a necesitar modificar por el momento.

Para más detalles: <http://www.mclarenx.com/2015/02/04/raspberry-pi-paso-2-primer-arranque-raspi-config/>

## 1. Configuración del teclado:

Empezaremos por **5. Internationalisation Options**, permite modificar la configuración del teclado al idioma deseado entre otras cosas.

**I1 Change local:** Para utilizar el español de España tendrás que elegir es\_ES.UTF8 UTF8

**I2 Change Timezone:** Aquí escogeremos Europa y después Madrid.

**I3 Change Keyboard Layout:** Elige el que se adapte al teclado que vayas a utilizar, pero en general seleccionando uno genérico en nuestro idioma debería ser suficiente. Además el “ayudante” pregunta varias cosas de teclas especiales, si no sabes por dónde van los tiros, déjalo por defecto.

## 2. Redimensión del disco duro:

En este punto hemos de recordar que el tamaño de nuestra tarjeta *SD* se redujo, para comprobar-lo desde aquí, podemos optar al comando **df -h** y ver que el % ocupado en la raíz es demasiado grande.

```
pi@raspberrypi:~$ df -h
S.ficheros      Tamaño Usados  Disp  Uso%  Montado en
/dev/root       1,3G   911M   282M   77%  /
devtmpfs        459M     0   459M    0%  /dev
tmpfs           463M     0   463M    0%  /dev/shm
tmpfs           463M   6,2M   457M    2%  /run
tmpfs           5,0M   4,0K   5,0M    1%  /run/lock
tmpfs           463M     0   463M    0%  /sys/fs/cgroup
/dev/mmcblk0p1  60M    20M    41M   34%  /boot
pi@raspberrypi:~$
```

Manual 1: Tamaño inicial del disco duro

Queremos recuperar la memoria, para ello entramos en el panel de configuración con **raspi-config** y seleccionamos la opción **1. Expand Filesystem**. Reiniciamos la máquina y ejecutamos de nuevo **df -h**, podemos observar que el tamaño disponible ahora es el de nuestra tarjeta.

```
The programs included with the Debian GNU/Linux system
the exact distribution terms for each program are desc
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to
permitted by applicable law.
pi@raspberrypi:~ $ df -h
S.ficheros      Tamaño Usados  Disp  Uso%  Montado en
/dev/root       15G    912M    14G   7%    /
devtmpfs        459M     0    459M   0%    /dev
tmpfs           463M     0    463M   0%    /dev/shm
tmpfs           463M    6,2M   457M   2%    /run
tmpfs           5,0M    4,0K   5,0M   1%    /run/lock
tmpfs           463M     0    463M   0%    /sys/fs/cgroup
/dev/mmcblk0p1  60M     20M    41M   34%   /boot
pi@raspberrypi:~ $
```

Manual 1: Tamaño redimensionado

### Configuración IP estática:

---

Para el proyecto necesitaremos configurar una *IP LAN* estática, de este modo conectando la máquina en otro entorno podremos acceder a ella fácilmente a través de su *IP*. Como podríamos hacer desde el comando **ssh** de *Linux* o un cliente como **Putty** para otras plataformas.

Si lanzamos la orden **ifconfig** nos aparecerán las *interfaces* de red; como vemos hay dos la **loopback** y la **eth0** (LAN/internet).

```
pi@raspberrypi:~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:f5:60:13
          inet addr:192.168.1.40  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::942b:b416:2b58:956a/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4210  errors:0  dropped:0  overruns:0  frame:0
          TX packets:2224  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:6074673 (5.7 MiB)  TX bytes:153002 (149.4 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Manual 1: Interfaces de red iniciales

3. Para declarar una *IP* estática para nuestra máquina, modificamos el fichero `/etc/network/interfaces`, solo nos centraremos en la línea `"iface eth0 inet manual"`, la modificaremos y añadiremos lo siguiente:

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.168.1.10 # Dirección a elegir del rango de tu router
```

```
netmask 255.255.255.0 # Rango.
```

```
gateway 192.168.1.1 # Dirección del router
```

4. Para cargar ahora nuestra nueva *IP*:

Bajamos la interface: `sudo ifdown eth0`

Subimos la interface: `sudo ifup eth0`

Reiniciamos el servicio: `sudo /etc/init.d/networking restart`.

Volvemos a ver que se ha asignado la *IP* que hemos definido con el comando `"ifconfig"`, de nuevo.

```
pi@raspberrypi:~$ sudo /etc/init.d/networking restart
[...] Restarting networking (via systemctl): networking.serviceWarning: Unit
le of networking.service changed on disk, 'systemctl daemon-reload' recommende
. ok
pi@raspberrypi:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:f5:60:13
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::942b:b416:2b58:956a/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:147 errors:0 dropped:0 overruns:0 frame:0
          TX packets:295 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14750 (14.4 KiB)  TX bytes:44128 (43.0 KiB)

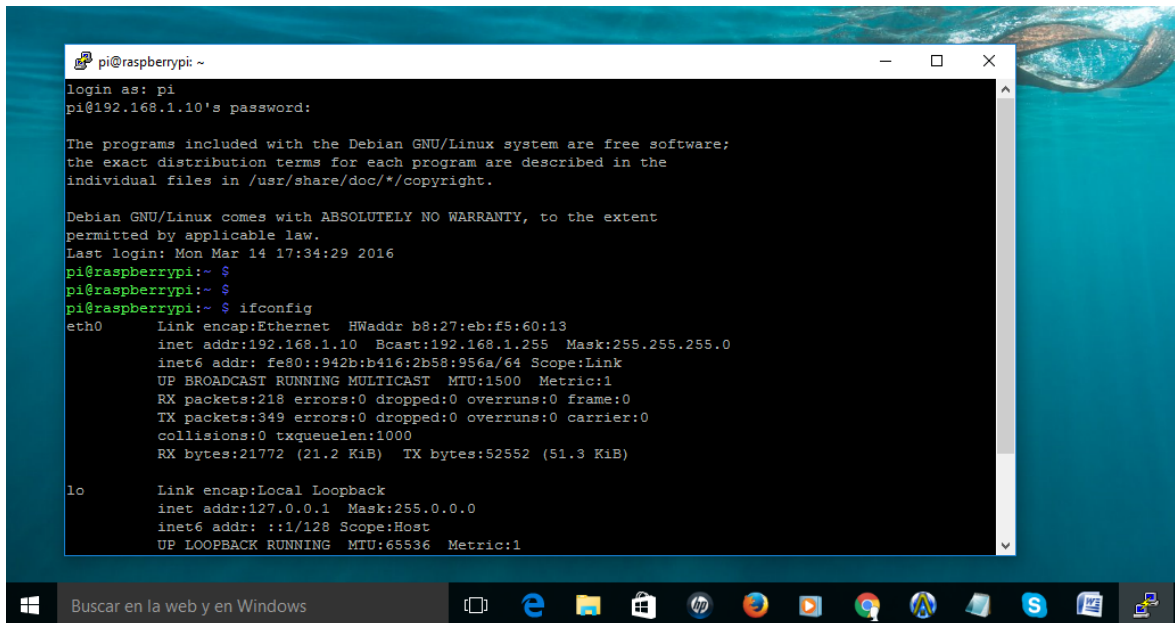
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

pi@raspberrypi:~$
```

Manual 1: *IP* estática configurada

Vamos a hacer la prueba de conectarnos desde otra máquina.

## Windows 10:



```
pi@raspberrypi: ~
login as: pi
pi@192.168.1.10's password:

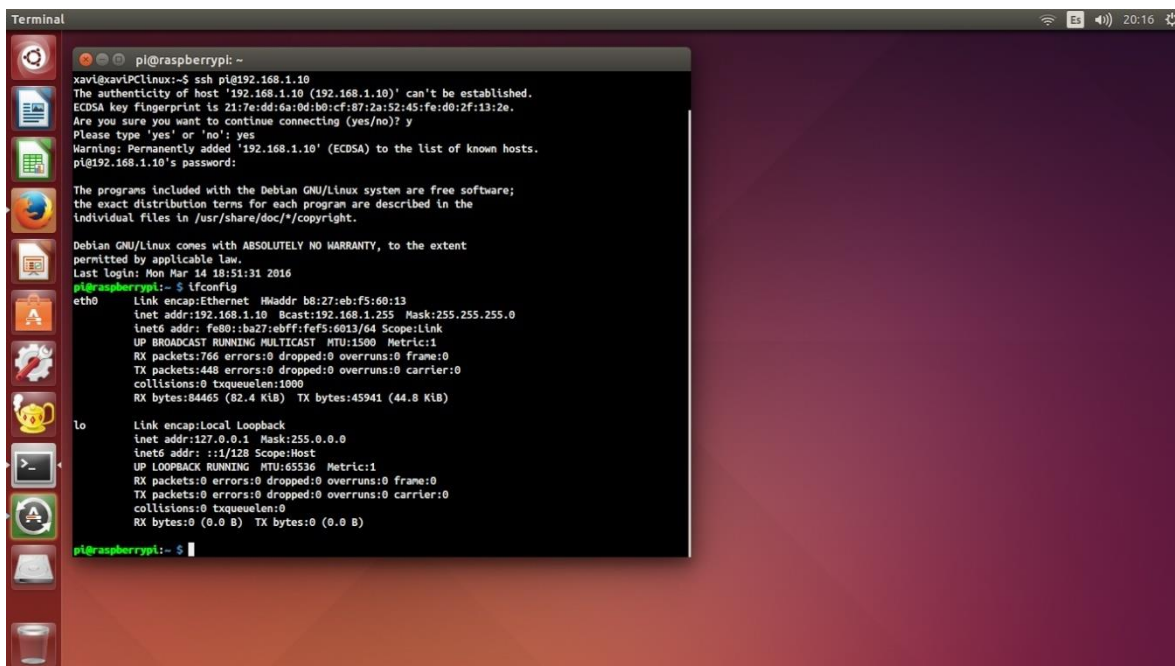
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar 14 17:34:29 2016
pi@raspberrypi:~$
pi@raspberrypi:~$
pi@raspberrypi:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:f5:60:13
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::942b:b416:2b58:956a/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:218  errors:0  dropped:0  overruns:0  frame:0
          TX packets:349  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:21772 (21.2 KiB)  TX bytes:52552 (51.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

Manual 1: Conexión *Windows – Raspberry Pi 2*

## Ubuntu 14.04



```
Terminal
pi@raspberrypi: ~
xavi@xaviPClinux:~$ ssh pi@192.168.1.10
The authenticity of host '192.168.1.10 (192.168.1.10)' can't be established.
ECDSA key fingerprint is 21:7e:dd:6a:0d:b0:cf:87:2a:52:45:fe:d0:2f:13:2e.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added '192.168.1.10' (ECDSA) to the list of known hosts.
pi@192.168.1.10's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar 14 18:51:31 2016
pi@raspberrypi:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:f5:60:13
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::ba27:ebff:fe:f5:6013/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:766  errors:0  dropped:0  overruns:0  frame:0
          TX packets:448  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:84465 (82.4 KiB)  TX bytes:45941 (44.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

pi@raspberrypi:~$
```

Manual 1: Conexión *Ubuntu 14.04 – Raspberry Pi 2*

## Instalación de QEMU Emulador:

---

Como concluimos anteriormente utilizaremos el *software Qemu* para emular arquitectura *x86* sobre *ARM*. Para descargarlo no utilizaré los archivos fuentes, sino que usaré los repositorios, pero antes es recomendable hacer un **apt-get update** para actualizarlos.

1. Instalamos **Qemu**: **sudo apt-get install qemu**

**Qemu** tiene múltiples opciones, no me centraré en explicarlas todas, solo usaré aquellas que sean útiles para esta primera implementación/emulación.

Mi objetivo en este punto es emular **Windows 98** sobre la *Raspberry Pi 2*.

Antes de todo obtengo los ficheros **.iso's** correspondientes y la clave necesaria para la instalación de *Windows 98*.

2. Creo un directorio para el sistema operativo y poder trabajar con más facilidad.

```
$ mkdir Win98SO
```

Dentro de la carpeta creada, guardaremos la **clave.txt** y el SO **windows98se.iso**.

3. Ahora ejecutamos el siguiente comando que nos creará el “disco/espacio” donde se encontrará nuestra máquina emulada con el SO.

```
$ qemu-img create WinImg98.img 8G (tamaño del disco duro emulado)
```

```
pi@raspberrypi:~/Win98SO $ ls
KEY.txt  windows98se.iso
pi@raspberrypi:~/Win98SO $ qemu-img create WinImg98.img 8G
Formatting 'WinImg98.img', fmt=raw size=8589934592
pi@raspberrypi:~/Win98SO $ ls
KEY.txt  windows98se.iso  WinImg98.img
pi@raspberrypi:~/Win98SO $ ls -la
total 640252
drwxr-xr-x 2 pi pi      4096 mar 14 18:23 .
drwxr-xr-x 3 pi pi      4096 mar 14 18:14 ..
-rw-r--r-- 1 pi pi       1042 mar 14 18:02 KEY.txt
-rw-r--r-- 1 pi pi  655601664 mar 14 18:07 windows98se.iso
-rw-r--r-- 1 pi pi  8589934592 mar 14 18:23 WinImg98.img
pi@raspberrypi:~/Win98SO $
```

Manual 1: Key, ISO y espacio de disco para *Windows 98*.

Esto nos generará el fichero **WinImg98.img**, o como lo hayamos definido.

4. El siguiente paso es indicar la arquitectura que queremos emular y el sistema operativo que vamos a instalar en el disco creado anteriormente.

Lanzamos el comando siguiente y empezará el proceso para la instalación de *Windows 98*.

```
$sudo qemu-system-x86_64 -boot d -hda WinImg98.img -cdrom windows98se.iso -m 512 -localtime -k es
```

Analizamos la sentencia:

**-boot d:** Indicamos que vamos a arrancar desde el CD(.iso)

**-hda WinImg98.img:** Indicamos que la imagen esta en disco duro, seguidamente indicamos la ruta de dicha imagen.

**-cdrom windows98se.iso:** Indicamos que vamos a instalar mediante CD y seguidamente le tenemos que decir la ruta del dispositivo donde se encuentra la .iso

**-m 512:** Cantidad de memoria virtual utilizada, es importante no colocar más cantidad de memoria de la que tenemos porque de lo contrario obtendremos un error indicándonos que no poseemos memoria suficiente.

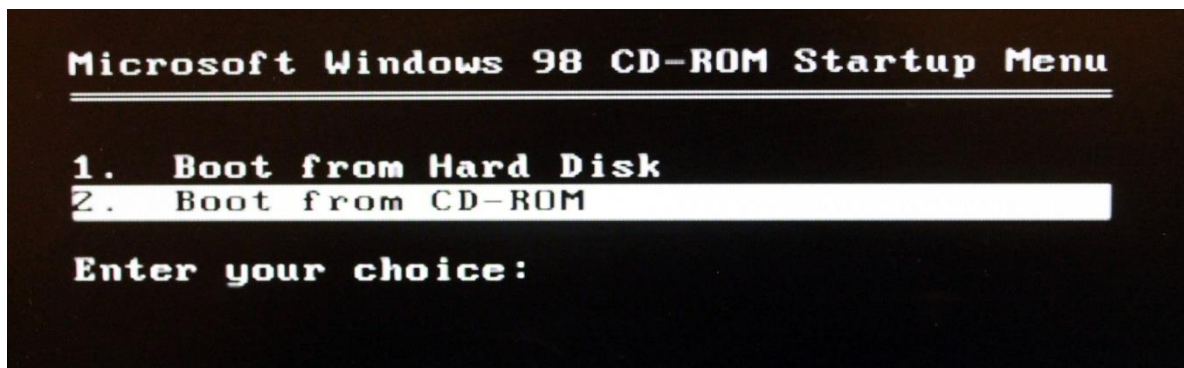
**-k es:** Especificamos el tipo de teclado que utilizaremos, “es” es equivalente a Español.

### **Instalación del Sistema Operativo “Inquilino”:**

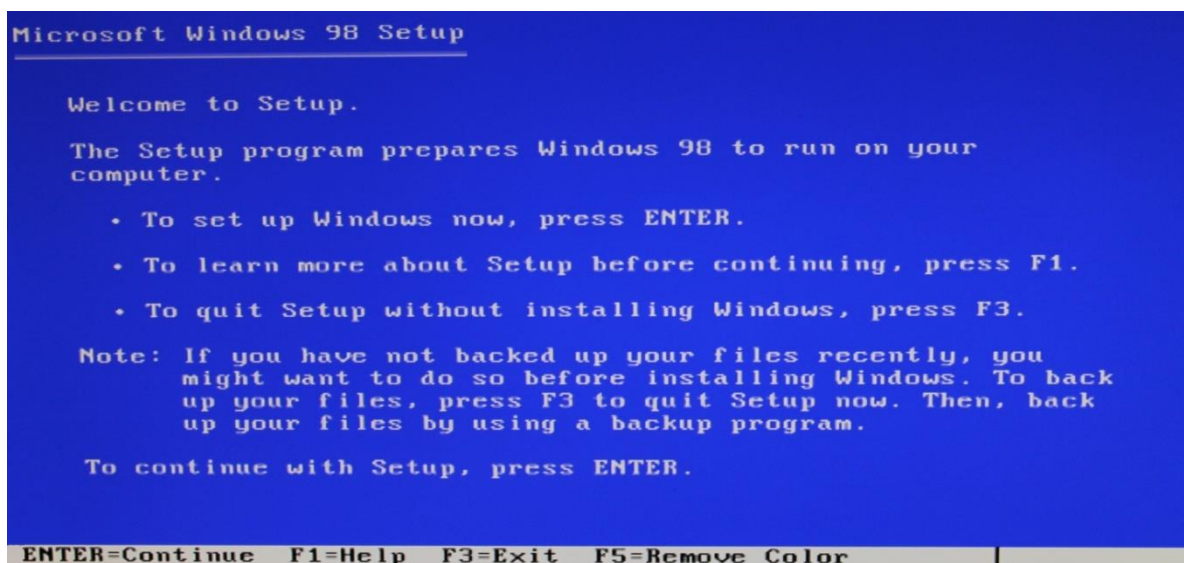
---

Llegados a este punto el proceso de instalación del sistema operativo es de la forma habitual. El propio instalador nos indicará los pasos a seguir para completar la tarea. Concretaré los pasos a seguir mediante las capturas.

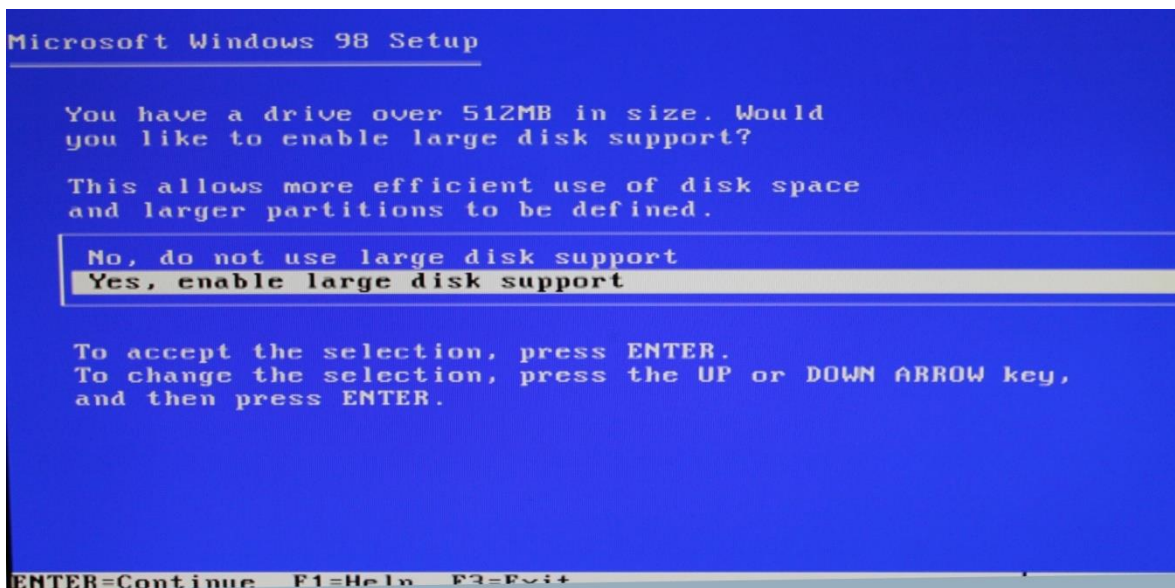




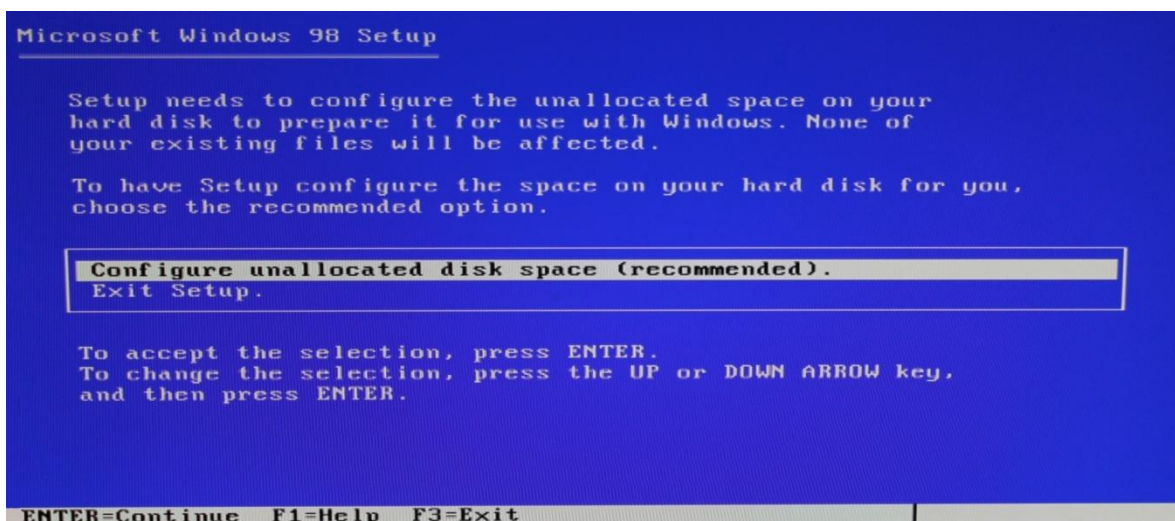
Manual 1: Arrancamos desde el CD, nuestra ISO.



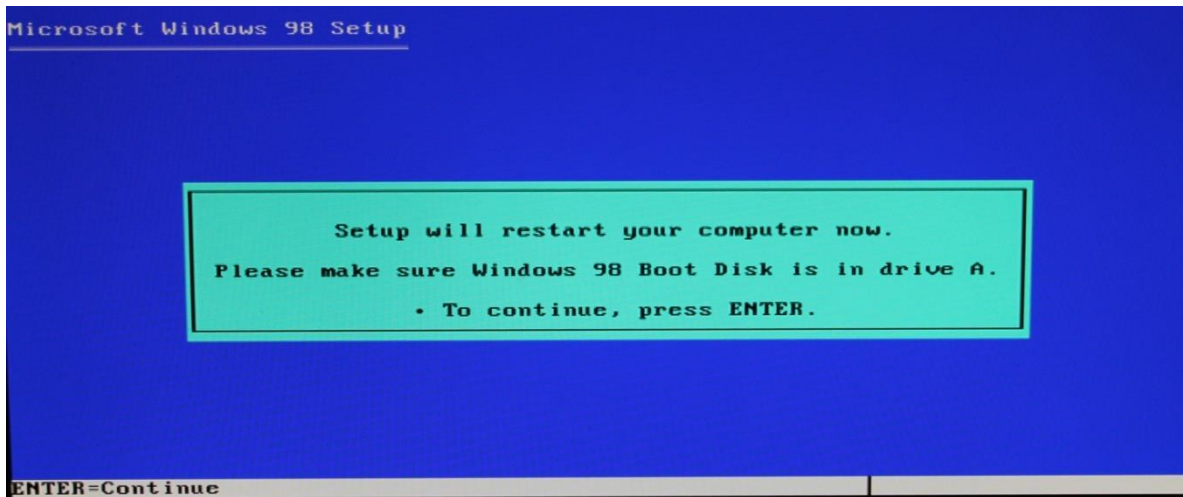
Manual 1: Captura de la pantalla de *Setup* de disco



Manual 1: Habilitamos el soporte para el disco.



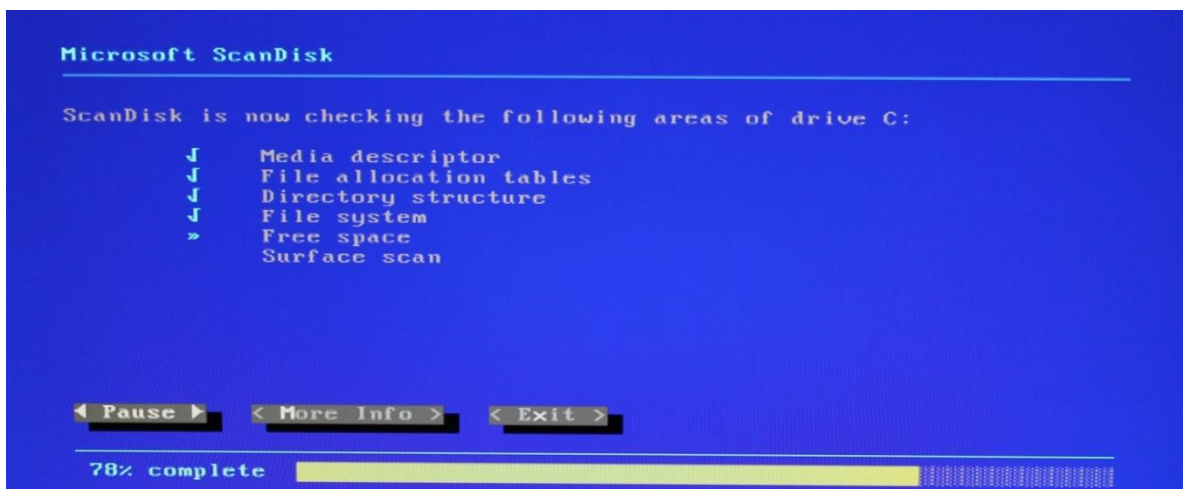
Manual 1: Configurar el espacio de disco sin asignar, opción recomendada.



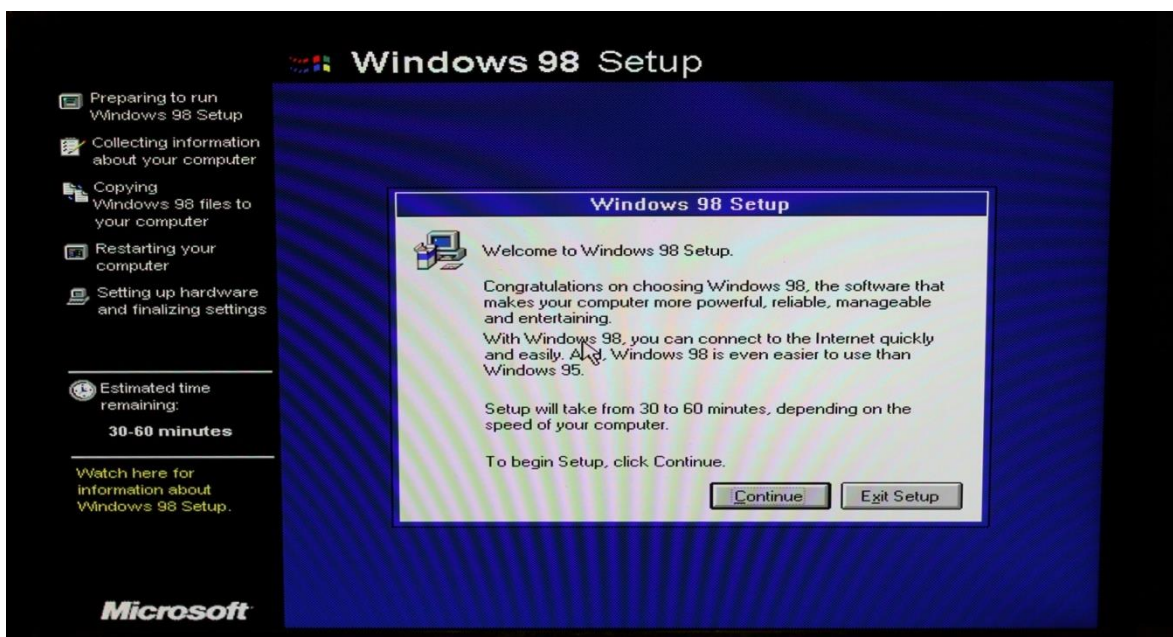
Manual 1: Reiniciamos el sistema.



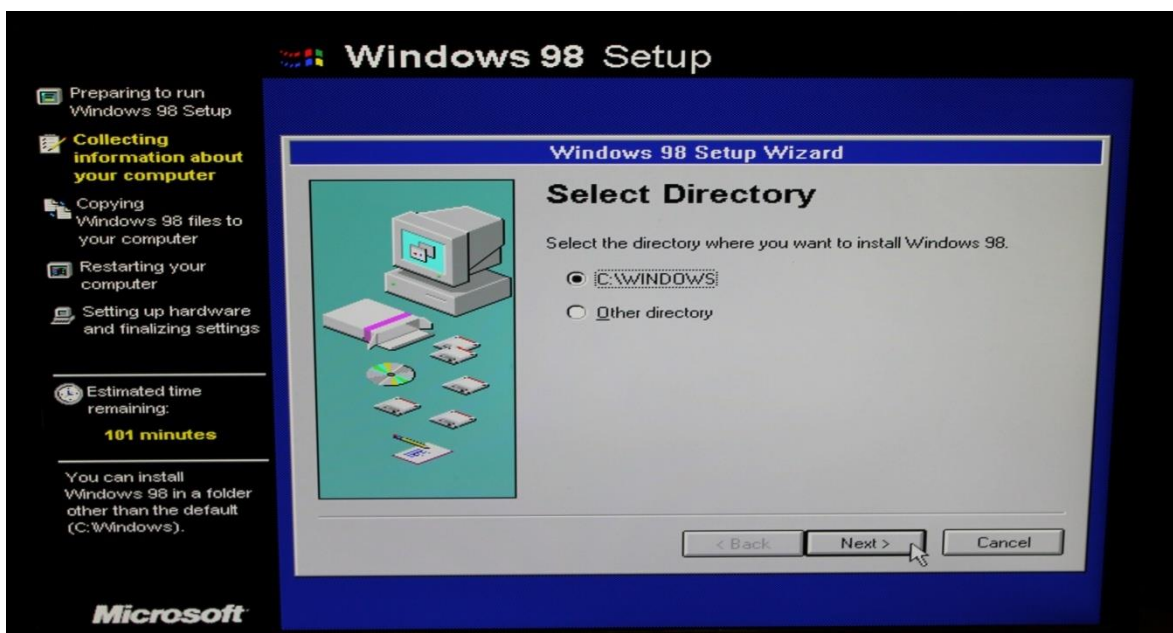
Manual 1: El disco duro se formatea.



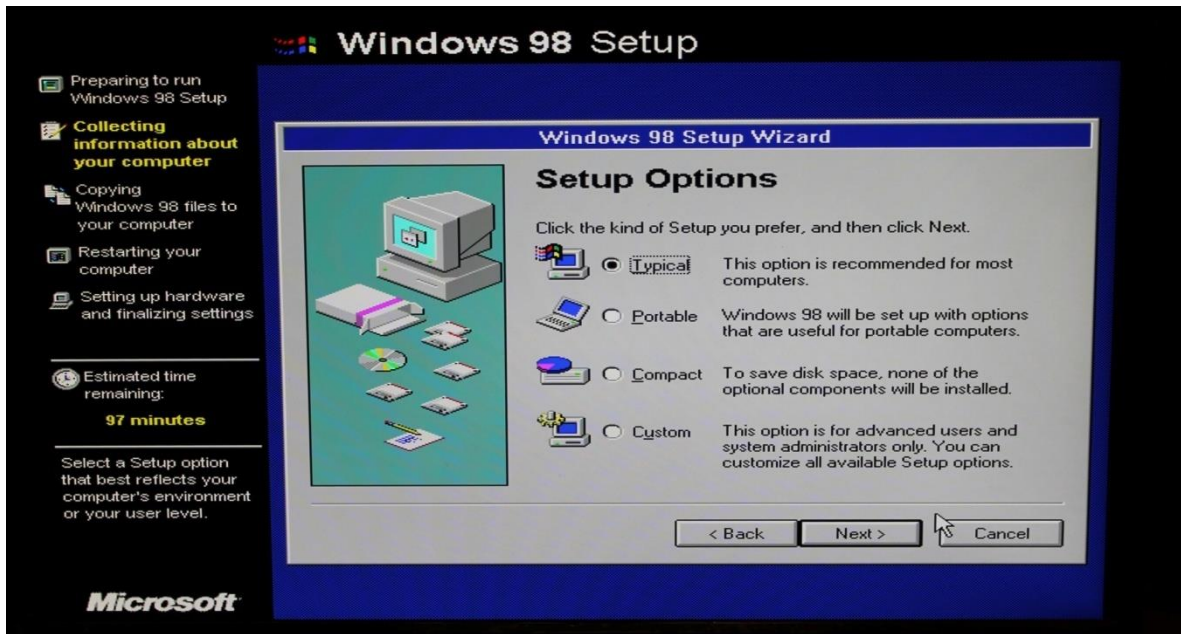
Manual 1: Escáner y *Checking* de las áreas del disco.



Manual 1: Interface para la instalación de *Windows*.



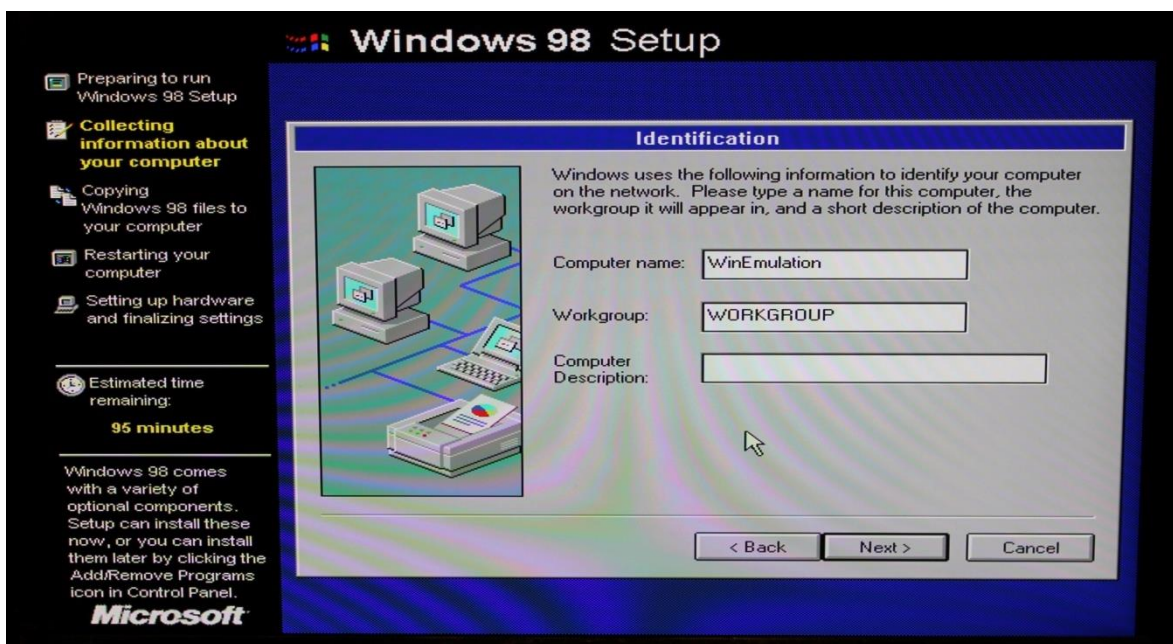
Manual 1: Ruta donde almacenar nuestro disco.



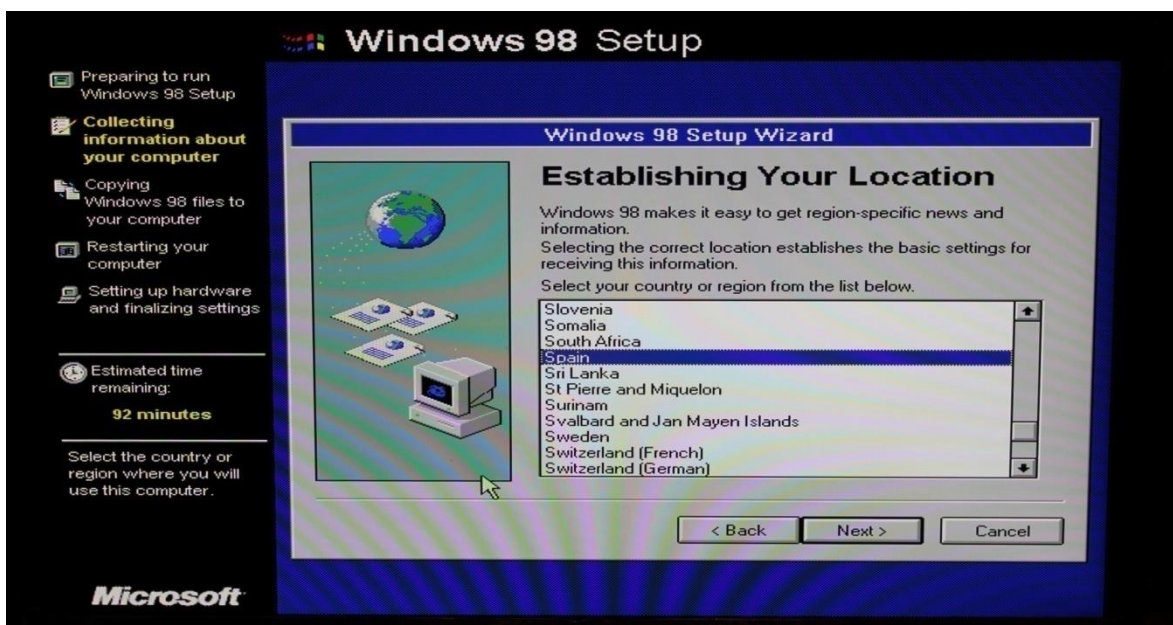
Manual 1: Tipo de instalación *Typical*.



Manual 1: Instalación de los componentes.



Manual 1: Nombre de la máquina.



Manual 1: Establecemos nuestra Ubicación/País.



Manual 1: La instalación empieza satisfactoriamente.



Manual 1: Reiniciamos la máquina

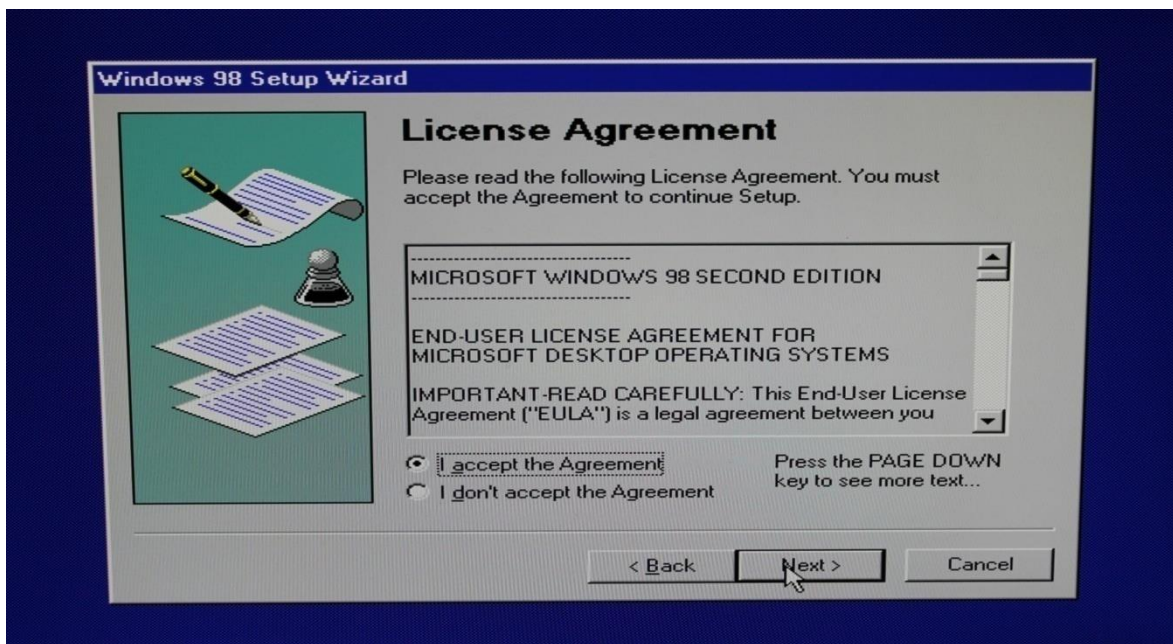


Manual 1: Logo de *Windows 98*, empieza la configuración básica.

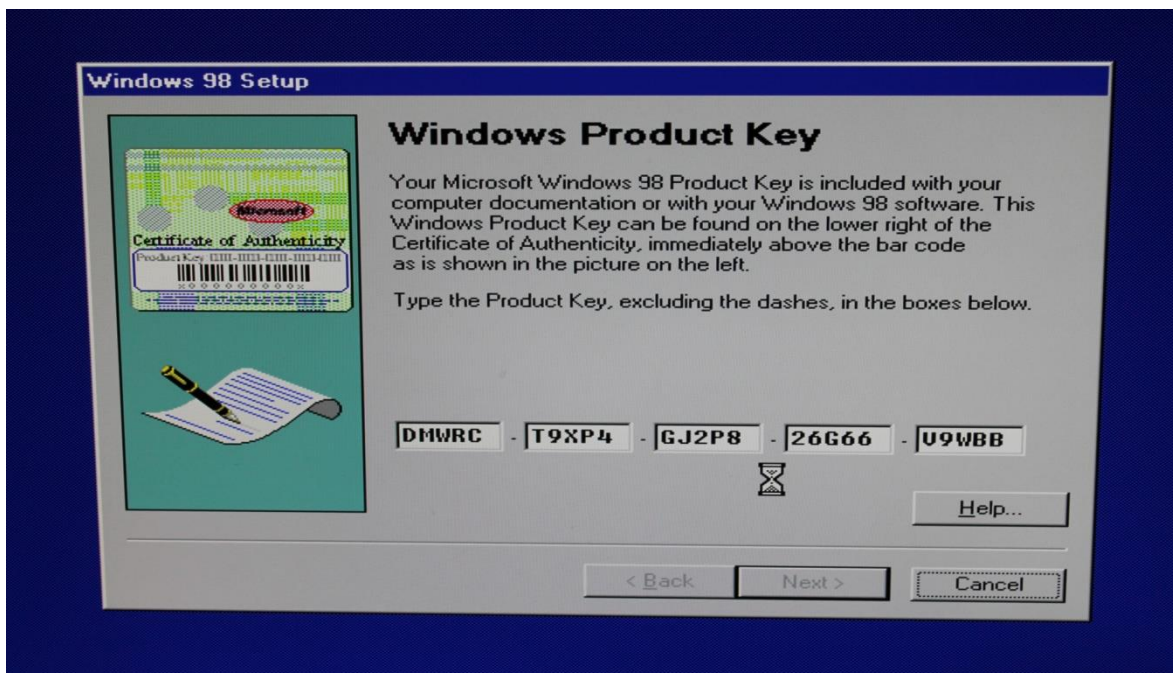


Manual 1: De nuevo la información del usuario

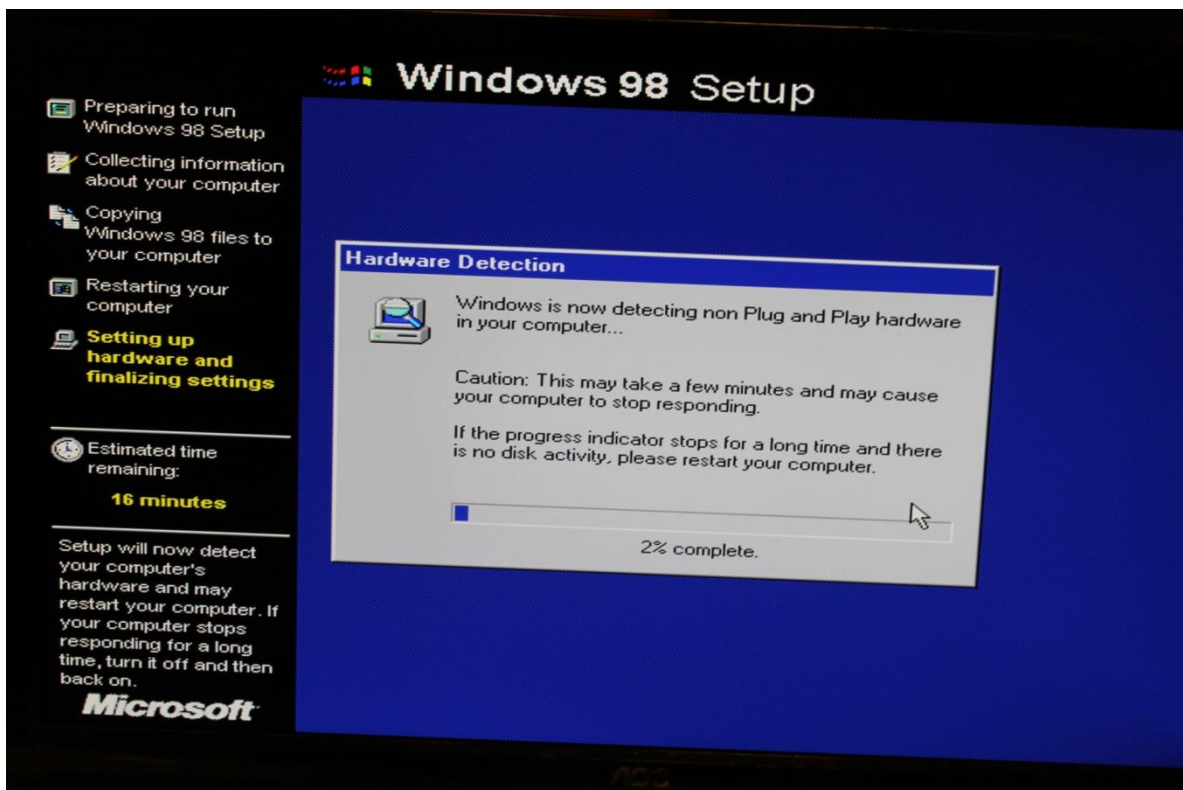




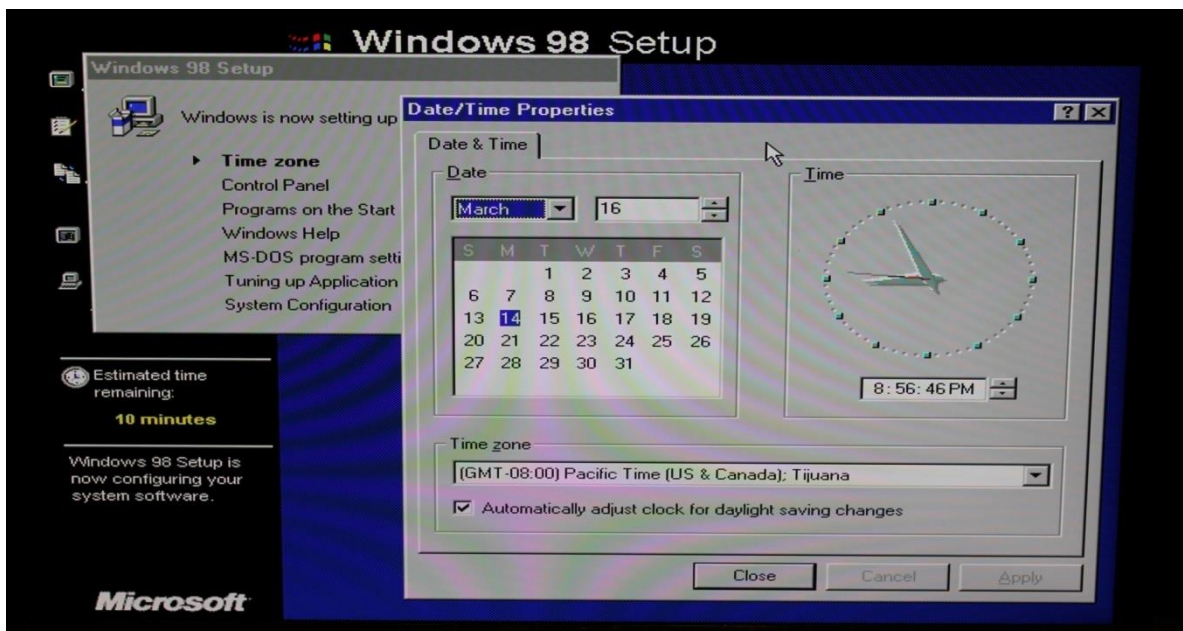
Manual 1: Aceptación los términos.



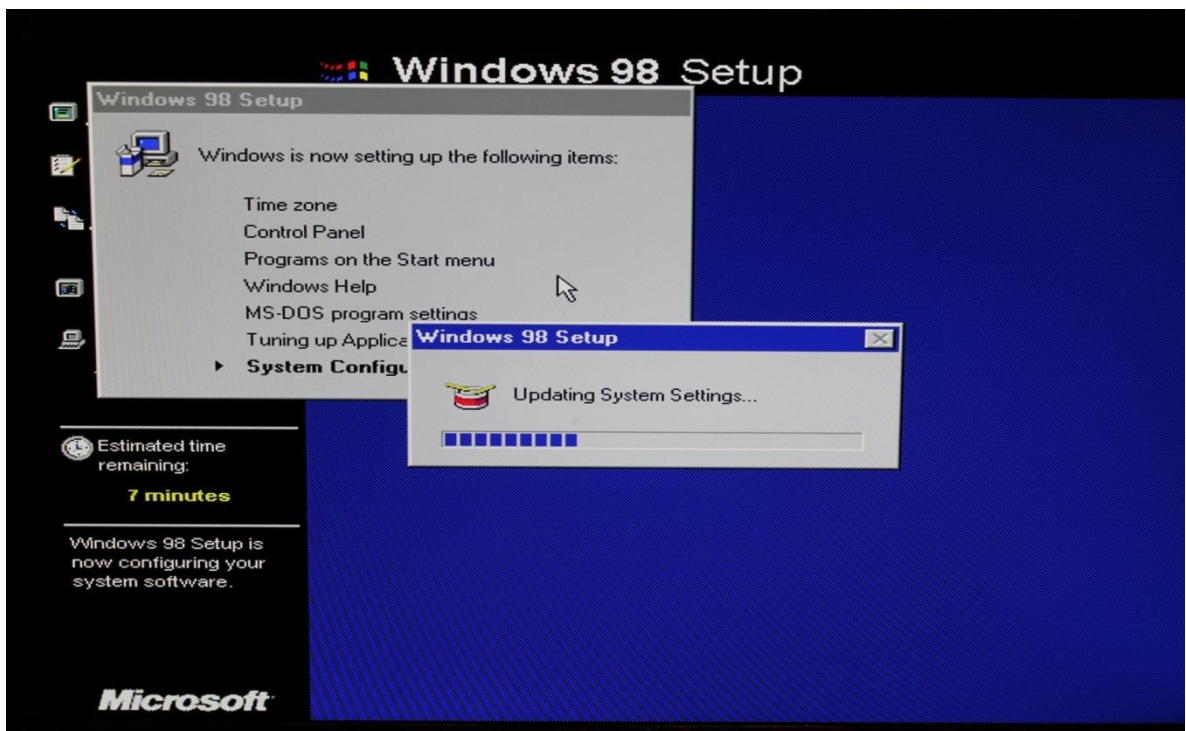
Manual 1: Añadimos la clave para Windows.



Manual 1: Detección de *plugins* y *hardware*.

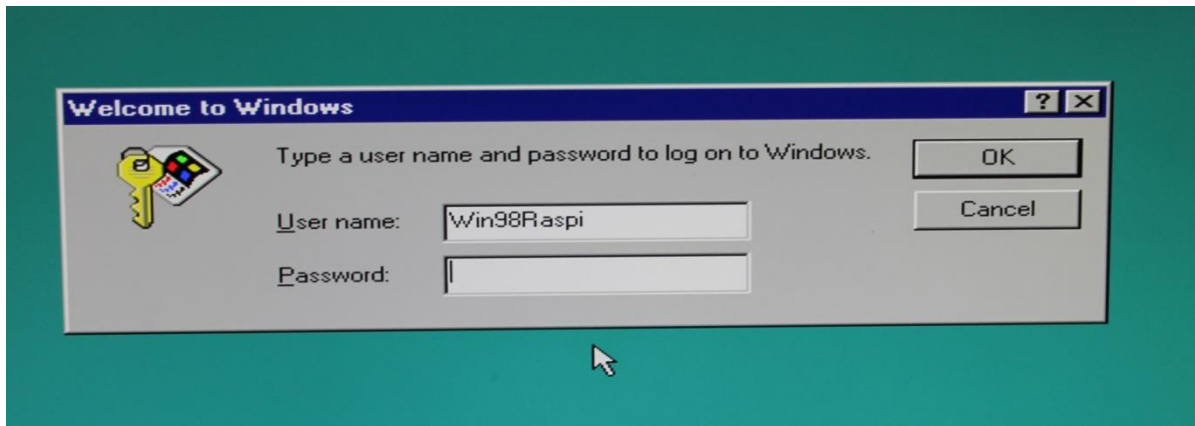


Manual 1: Configuración de la hora.



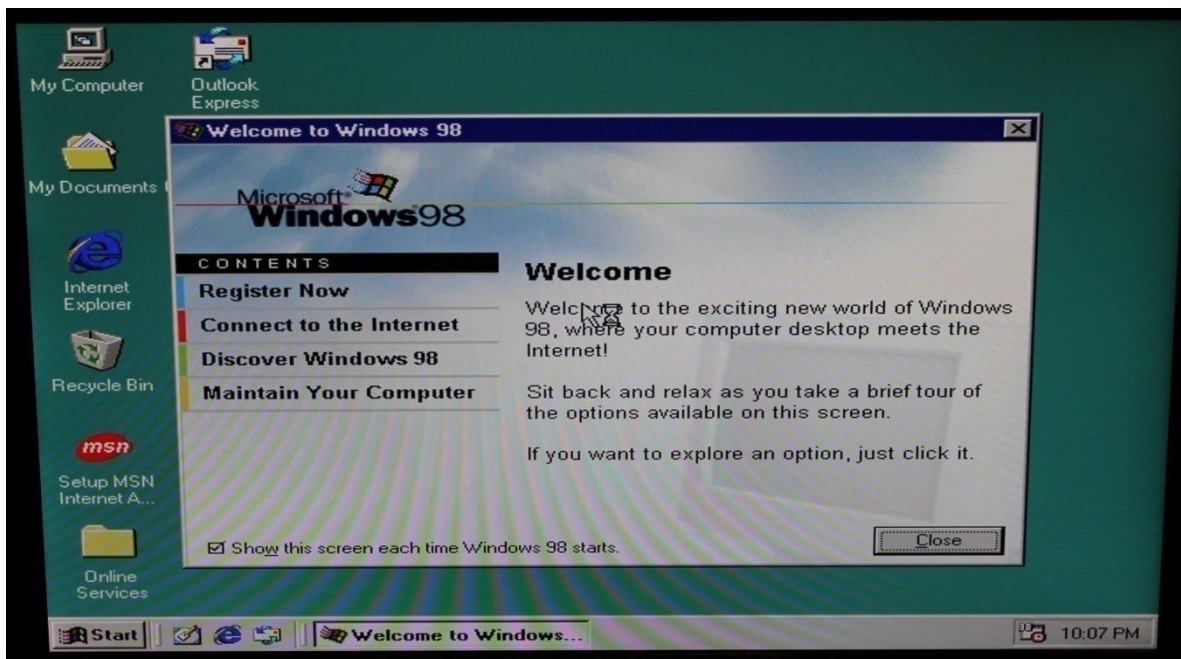
Manual 1: Actualizando las configuraciones de *Windows*.

## Resultado



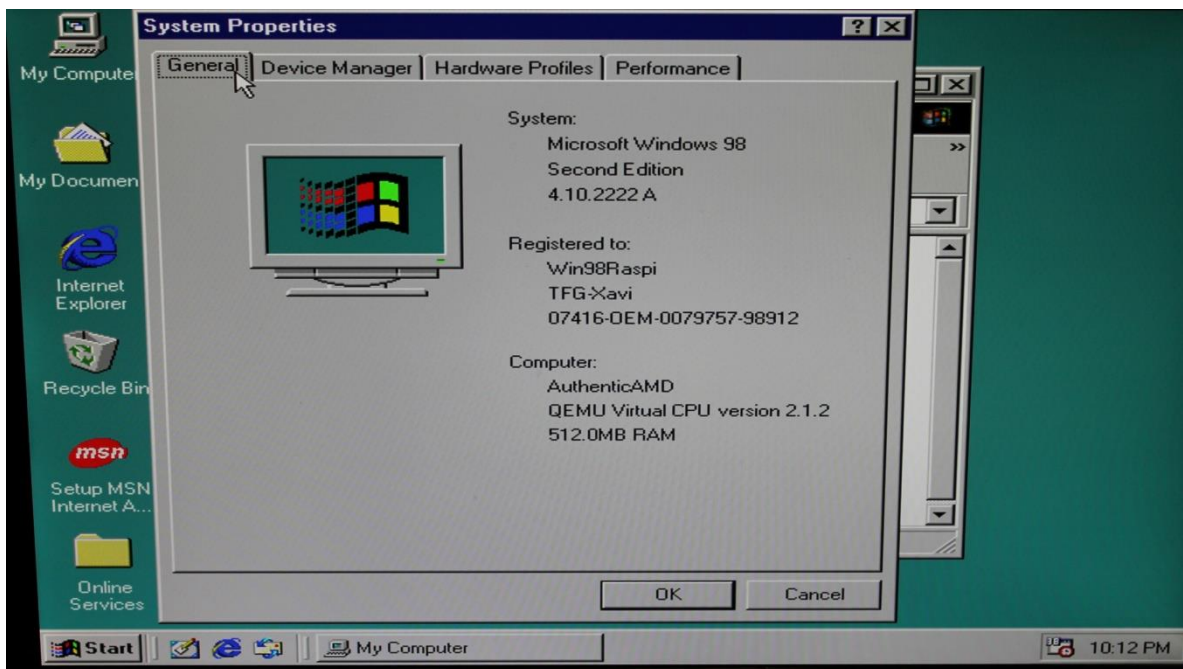
Manual 1: Login de entrada al SO.

Finalmente accedemos al Escritorio de *Windows* y podemos trabajar sobre este sistema operativo desde nuestra *Raspberry Pi 2*.

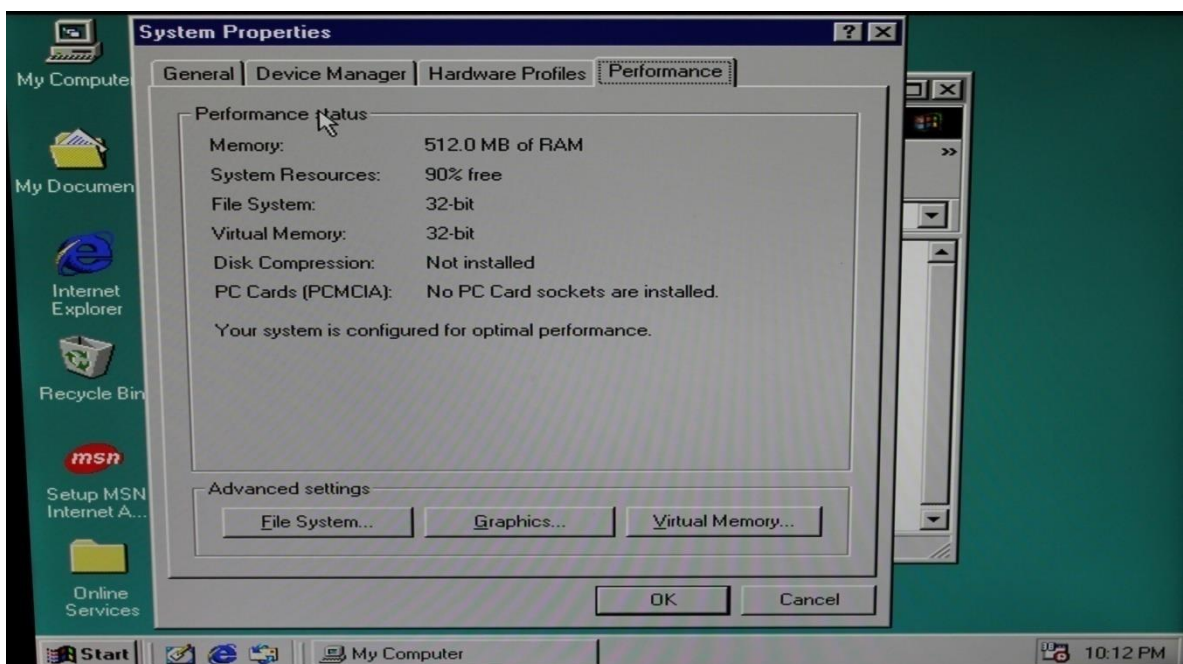


Manual 1: Escritorio de Windows 98.

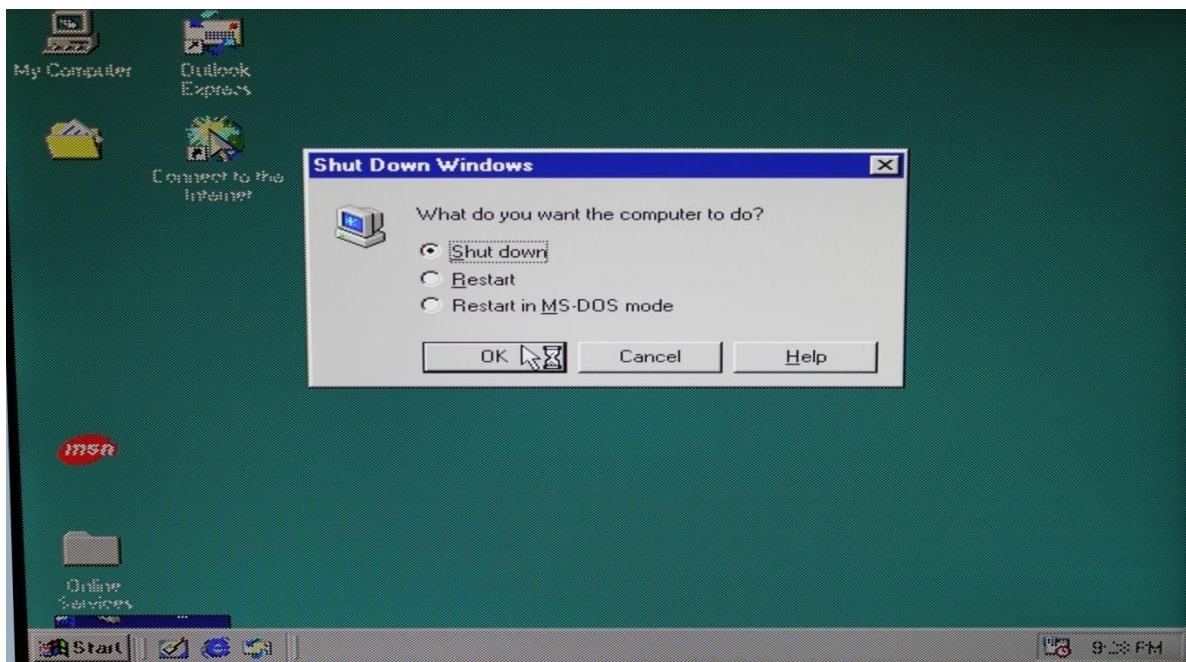
En las propiedades del sistema podemos ver que hemos emulado la arquitectura x86 del Sistema Operativo *Windows 98* con *Qemu*, y si nos fijamos en la memoria *RAM* corresponde a la definida en el comando de instalación.



Manual 1: Propiedades del Sistema – General.



Manual 1: Propiedades del Sistema – Performance



Manual 1: Cerramos Windows 98.

Una vez hemos cerrado *Windows* para volver a encenderlo usaremos el siguiente comando:

```
$sudo qemu-system-x86_64 -boot d -hda WinImg98.img -m 512
```

## ANEXO B:

### MANUAL NUMERO 2 OPENSTACK

---

#### Requisitos Hardware

- 2 RASPBERRYS
- CABLES DE RED
- FUENTE DE ALIMENTACIÓN
- (Opcional) PANTALLA I TECLADO
- DISCO DURO (tarjeta SD 8G mínimo)

#### Requisitos Software

- SISTEMA OPERATIVO (*Ubuntu Trusty Tahr 14.04 LTS*)
- IP ESTÁTICA (Configuración en MANUAL 1 Anexo A)
- ASIGNACIÓN DE HOSTNAME (Configuración en MANUAL Anexo A)
- EXPANSION DEL SISTEMA (Configuración en MANUAL Anexo A)
- COMENTAR LA LINEA CON EL *HOST 127.0.1.1 xxxxx* DEL FILE */etc/hosts*

### INSTALACIÓN OPENSTACK Ubuntu Xenial Xerus 16.04 LTS

---

Los siguientes pasos serán para la configuración e instalación de los módulos básicos para el nodo *RASPBERRY CONTROLLER*.

#### Gestor base de datos *MYSQL*

---

1. Instalación de los paquetes necesarios para *MySQL*.

```
# apt-get install python-pymysql
```

```
# apt-get install mysql-server
```

2. Nos pide añadir una password para el usuario root del Gestor de la Base de Datos.

En mi caso puse: ubuntu.

### 3. Opciones de configuración

En la guía oficial para instalar *Openstack* con los módulos de la versión de Liberty nos dice que se debe modificar el siguiente fichero para la configuración de MySQL.

```
# vim /etc/mysql/conf.d/mysqld_openstack.cnf
```

En la placa *Raspberry* ese fichero está en otra ubicación así que el archivo que se debe modificar es el siguiente:

```
# vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

### 4. Finalmente reiniciamos el servicio e iniciamos el script para la securización.

```
# service mysql restart
```

```
# mysql_secure_installation
```

 al ejecutar este comando nos hace las siguientes preguntas.

*Change the password for root ? ((Press y|Y for Yes, any other key for No) : n*

*Remove anonymous users? (Press y|Y for Yes, any other key for No) : y*

*Disallow root login remotely? (Press y|Y for Yes, any other key for No) : n*

*Remove test database and access to it? (Press y|Y for Yes, any other key for No) : y*

*Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y*

## Sistema de colas de mensaje servicio *RABBITMQ*

---

### 1. Instalación de los paquetes necesarios para *RABBITMQ*

```
# apt-get install rabbitmq-server
```

### 2. Necesitamos un usuario principal para el acceso a este servicio, en este caso el usuario será *openstack* con *password* ubuntu.

```
# rabbitmqctl add_user openstack 'RABBIT_PASS=ubuntu'
```



3. Damos permisos de acceso, lectura y escritura sobre la configuración de RABBITMQ para el usuario creado, esto es necesario porque este servicio será llamado desde otros módulos de manera automática.

```
# rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

### Módulo de autenticación *KEYSTONE*

---

1. Creamos la Base de datos para el módulo de *Keystone*. Recordemos que cada modulo de *Openstack* tiene su propia base de datos. La base de datos se ha llamado *keystone*.

```
# mysql -u root -p
```

```
mysql> CREATE DATABASE keystone;
```

```
Query OK, 1 row affected (0.00 sec)
```

2. Creamos el usuario con permisos sobre la BD keystone. El usuario se ha llamado keystone con la password 'keystone\_DBPASS\_0'.

```
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'  
IDENTIFIED BY 'keystone_DBPASS_0';
```

```
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'  
IDENTIFIED BY 'keystone_DBPASS_0';
```

```
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> exit
```

3. Generamos un número por defecto que será la clave/*token* de administración para la configuración inicial de keystone.

```
# openssl rand -hex 10
```

```
c00c155c2fe0ef94646a
```

4. Deshabilitar el servicio al iniciar automáticamente el servicio después de la instalación.

```
root@ubuntu-controller:~# echo "manual" > /etc/init/keystone.override
```

5. Instalación de los paquetes necesarios para *Keystone*

```
# apt-get install keystone  
# apt-get install apache2  
# apt-get install libapache2-mod-wsgi  
# apt-get install memcached  
# apt-get install python-memcache
```

6. Modificamos fichero de configuración de *Keystone*

```
# vim /etc/keystone/keystone.conf
```

7. Llenamos automáticamente la base de datos de *keystone* con las tablas que se necesitan, esto tarda unos minutos.

```
# /bin/sh -c "keystone-manage db_sync" keystone
```

8. Modificamos la configuración de apache para definir el nodo controlador que tendrá el servicio *Keystone* instalado.

```
# vim /etc/apache2/apache2.conf
```

9. Añadimos un nuevo fichero en *apache* para definir que al servicio de *Keystone* se accederá mediante los puertos 5000 y 35357 de manera virtual.

```
# vim /etc/apache2/sites-available/wsgi-keystone.conf
```

10. Habilitamos el fichero creado en el paso anterior.

```
# ln -s /etc/apache2/sites-available/wsgi-keystone.conf /etc/apache2/sites-enabled
```

11. Reiniciamos el servicio *apache*.

```
# service apache2 restart
```

12. Por defecto al instalar el modulo de *Keystone*, se crea una BD *SQLite* que no vamos a utilizar, así que la podemos eliminar sin problemas.

```
# rm -rf /var/lib/keystone/keystone.sqlite
```

13. *Keystone* no contiene de momento ningún servicio capaz de gestionar la autenticación de usuarios. Así que debemos crear ese servicio, primero configuramos las variables de entorno con los siguientes parámetros. El *token* o clave generada anteriormente, la URL de acceso para *Keystone* y la versión de la *API*.

```
# export OS_TOKEN=c00c155c2fe0ef94646a
```

```
# export OS_URL=http://192.168.1.50:35357/v3
```

```
# export OS_IDENTITY_API_VERSION=3
```

14. El servicio de identidad gestiona un catálogo de servicios en el entorno de *OpenStack*. Primero vamos a crear este servicio.

```
# openstack service create --name keystone --description "OpenStack Identity" identity
```

15. *OpenStack* utiliza tres variantes *API* de punto final para cada servicio: admin, interna y pública. El punto final de la *API* de administración permite la modificación de los usuarios y los inquilinos por defecto, mientras que las *API* públicas e internas no permiten estas operaciones. En un entorno de producción, las variantes pueden residir en redes separadas que dan servicio a diferentes tipos de usuarios por razones de seguridad. Por ejemplo, la red *API* pública podría ser visible a través de Internet para que los clientes pueden gestionar sus nubes. La red *API* de administración puede estar restringida a los operadores dentro de la organización que gestiona la infraestructura. La red interna de la *API* podría estar restringido a los anfitriones que contienen los

servicios de *OpenStack*. Además, *OpenStack* es compatible con múltiples regiones para la escalabilidad.

En este caso creamos los 3 distintos puntos finales, ya que cada servicio que se agrega al entorno, requiere las tres variantes de punto final de la *API* en el servicio de identidad

```
# openstack endpoint create --region RegionOne identity public
```

```
http://192.168.1.50:5000/v2.0
```

```
# openstack endpoint create --region RegionOne identity internal
```

```
http://192.168.1.50:5000/v2.0
```

```
# openstack endpoint create --region RegionOne identity admin
```

```
http://192.168.1.50:35357/v2.0
```

16. Se crea un proyecto de administración, un usuario de admin, y se le agrega el rol correspondiente.

```
# openstack project create --domain default --description "Admin Project" admin
```

```
No domain with a name or ID of 'default' exists.
```

```
# openstack project create --domain=default --description "Admin Project" admin
```

```
No domain with a name or ID of 'default' exists.
```

```
# openstack project create --description "Admin Project" admin
```

```
You have tried to create a resource using the admin token. As this token is not within a domain you must explicitly include a domain for this resource to belong to. (HTTP 400) (Request-ID: req-2471e77d-b0df-4a21-85b5-d0fc1d837ef9)
```

Como vemos, siguiendo la guía oficial, los comandos para la creación de un proyecto, un usuario y una asignación de rol ha fallado, para solucionarlo se han cambiado las variables de entorno. Se ha modificado la versión de la URL para *Keystone* y se ha borrado la variable de entorno con la versión de la *API*.

```
# export OS_URL=http://192.168.1.50:35357/v2.0
```

```
# unset OS_IDENTITY_API_VERSION
```

```
# openstack project create --description "Admin Project" admin
```

```
# openstack user create --password-prompt admin
```

```
# openstack role create admin
```

```
# openstack role add --project admin --user admin admin
```

17. Se crea un proyecto de demo, un usuario de demo, y se le agrega el rol correspondiente.

```
# openstack project create --description "Service Project" service
```

```
# openstack project create --description "Demo Project" demo
```

```
# openstack user create --password-prompt demo
```

```
# openstack role create user
```

```
# openstack role add --project demo --user demo user
```

18. Podemos comprobar que todo ha salido correctamente ejecutando los siguientes comandos.

(Primero borramos las variables de entorno generadas) **# unset OS\_TOKEN OS\_URL**

```
# openstack --os-auth-url http://192.168.1.50:35357/v3 --os-project-domain-id default --os-user-domain-id default --os-project-name admin --os-username admin --os-auth-type password token issue
```

```
# openstack --os-auth-url http://192.168.1.50:5000/v3 --os-project-domain-id default --os-user-domain-id default --os-project-name demo --os-username demo --os-auth-type password token issue
```

19. Para realizar ciertas operaciones dependiendo del usuario sobre *openstack* podemos construir dos ficheros, uno para admin y otro para demo con las variables de entorno correspondientes.

```
# vim admin-openrc.sh
```

```
export OS_PROJECT_DOMAIN_ID=default
```

```
export OS_USER_DOMAIN_ID=default
```

```
export OS_PROJECT_NAME=admin
```

```
export OS_TENANT_NAME=admin
```

```
export OS_USERNAME=admin
```

```
export OS_PASSWORD=admin  
export OS_AUTH_UTL=http://192.168.1.50:35357/v3  
export OS_IDENTITY_API_VERSION=3
```

#### **# vim demo-openrc.sh**

```
export OS_PROJECT_DOMAIN_ID=default  
export OS_USER_DOMAIN_ID=default  
export OS_PROJECT_NAME=demo  
export OS_TENANT_NAME= demo  
export OS_USERNAME= demo  
export OS_PASSWORD= demo  
export OS_AUTH_UTL=http://192.168.1.50:5000/v3  
export OS_IDENTITY_API_VERSION=3
```

20. Para utilizar los clientes en este caso admin o demo solo se debe ejecutar el script y pedir un *token* de autenticación.

#### **# source admin-openrc.sh**

```
# openstack token issue
```

#### **# source demo-openrc.sh**

```
# openstack token issue
```

Podemos comprobar también que ha salido todo correcto ya que al crear el *token* aparecen los mismos valores de user\_id y project\_id que aparecieron en el momento de la creación del usuario y su respectivo proyecto.

### **Módulo gestor de imagenes *GLANCE***

---

Como en los demás módulos se necesita previamente crear una base de datos.

21. Creamos la base de datos para el modulo de *Glance*. La base de datos se ha llamado glance.

```
# mysql -u root -p
```

**Enter password:**

```
mysql> CREATE DATABASE glance;
```

```
Query OK, 1 row affected (0.00 sec)
```

22. Creamos el usuario con permisos sobre la BD glance. El usuario se ha llamado glance con la password 'glance\_DBPASS\_0'.

```
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost'  
IDENTIFIED BY 'glance_DBPASS_0';
```

```
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY  
'glance_DBPASS_0';
```

```
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> exit
```

**Bye**

23. Cargamos la configuración para el usuario admin.

```
# source admin-openrc.sh
```

24. Se crea un usuario llamado glance, y se le agrega el rol correspondiente.

```
# openstack user create --domain default --password-prompt glance
```

```
User Password: (ubuntu)
```

```
Repeat User Password: (ubuntu)
```

```
# openstack role add --project service --user glance admin
```

25. Se crea el servicio de identificación para *Glance*

```
# openstack service create --name glance --description "OpenStack Image  
service" image
```

26. Creación de los tres puntos finales para la *API*, public, internal y admin.

```
# openstack endpoint create --region RegionOne image public
```

```
http://192.168.1.50:9292
```

```
# openstack endpoint create --region RegionOne image internal
```

```
http://192.168.1.50:9292
```

```
# openstack endpoint create --region RegionOne image admin
```

```
http://192.168.1.50:9292
```

27. Instalación de los paquetes necesarios para *Glance*

```
# apt-get install glance
```

```
# apt-get install python-glanceclient
```

28. Se modifican y configuran los siguientes ficheros de configuración del módulo.

```
# vim /etc/glance/glance-api.conf
```

```
# vim /etc/glance/glance-registry.conf
```

29. Reiniciamos los servicios de Glance

```
# service glance-api restart
```

```
# service glance-registry restart
```

30. Llenamos automáticamente la base de datos de *Glance* con las tablas que se necesitan, esto tarda unos minutos.

```
# /bin/sh -c "glance-manage db_sync" glance
```

31. Por defecto al instalar el modulo de *Glance*, se crea una *BD SQLite* que no vamos a utilizar, así que la podemos eliminar sin problemas.

```
# rm -rf /var/lib/glance/glance.sqlite
```



32. Para comprobar que todo ha salido correctamente podemos descargarnos una imagen muy simple de apenas *13MB*. Para la configuración del modulo *Glance* es necesario usar la versión de la *API 2.0*.

```
# source admin-openrc.sh
# export OS_IMAGE_API_VERSION=2
# wget http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86\_64-disk.img
```

33. Añadimos la imagen al servicio de imágenes de *OpenStack*.

```
# glance image-create --name "cirros" --file cirros-0.3.4-x86_64-disk.img --disk-format qcow2 --container-format bare --visibility public --progress
```

34. Podemos ver las imágenes disponibles por terminal con el siguiente comando

```
# glance image-list
```

### Módulo gestor de máquinas virtuales *NOVA*

---

35. Creamos la base de datos para el módulo de *Nova*. La base de datos se ha llamado *nova*.

```
# mysql -u root -p
```

```
mysql> CREATE DATABASE nova;
Query OK, 1 row affected (0.00 sec)
```

36. Creamos el usuario con permisos sobre la BD *nova*. El usuario se ha llamado *nova* con la *password* `'nova_DBPASS_0'`

```
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' IDENTIFIED
BY 'nova_DBPASS_0';
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED BY
```

```
'nova_DBPASS_0';
```

```
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> exit
```

```
Bye
```

37. Cargamos la configuración para el usuario admin.

```
# source admin-openrc.sh
```

38. Se crea un usuario de nombre nova, y se le agrega el rol correspondiente.

```
# openstack user create --domain default --password-prompt nova
```

```
User Password: (ubuntu)
```

```
Repeat User Password: (ubuntu)
```

```
# openstack role add --project service --user nova admin
```

39. Se crea el servicio de identificación para Nova

```
# openstack service create --name nova --description "Openstack Compute"
compute
```

40. Creación de los tres puntos finales para la API, public, internal y admin.

```
# openstack endpoint create --region RegionOne compute public
```

```
http://192.168.1.50:8774/v2/%\(tenant\_id\)s
```

```
# openstack endpoint create --region RegionOne compute internal
```

```
http://192.168.1.50:8774/v2/%\(tenant\_id\)s
```

```
# openstack endpoint create --region RegionOne compute admin
```

```
http://192.168.1.50:8774/v2/%\(tenant\_id\)s
```

41. Instalación de los paquetes necesarios para Nova

```
# apt-get install nova-api
```

```
# apt-get install nova-cert
```

```
# apt-get install nova-consoleauth
```

```
# apt-get install nova-scheduler
```

```
# apt-get install nova-conductor
```

42. Se modifica y configura los siguientes ficheros de configuración del módulo.

```
# vim /etc/nova/nova.conf
```

43. Llenamos automáticamente la base de datos de *Nova* con las tablas que se necesitan, esto tarda unos minutos.

```
# /bin/sh -c "nova-manage db_sync" nova
```

44. Por defecto al instalar el modulo de Nova, se crea una *BD SQLite* que no vamos a utilizar, así que la podemos eliminar sin problemas.

```
# rm -rf /var/lib/nova/nova.sqlite
```

45. Reiniciamos los servicios

```
# service nova-api restart
```

```
# service nova-cert restart
```

```
# service nova-consoleauth restart
```

```
# service nova-scheduler restart
```

```
# service nova-conductor restart
```

```
# service nova-novncproxy restart
```

### Módulo para la interfaz web DASHBOARD.

---

46. Instalación de los paquetes necesarios para *Dashboard*.

```
#apt-get install openstack-dashboard
```

47. Se modifica y configura el fichero de configuración del módulo.

```
# vim /etc/openstack-dashboard/local_settings.py
```

48. Reiniciamos el servicio de apache

```
# service apache2 reload
```

En este punto ya se han instalado los módulos y servicios básicos de *OpenStack* a nuestra máquina *Raspberry Pi 2 MASTER*. Ahora simplemente vamos a añadir un nuevo nodo al sistema distribuido *cloud* que hemos generado.

### **RASPBERRY NODO**

Al NODO *Raspberry Pi* solo hace falta instalarle dos paquetes, configurar los ficheros y ya podrá acceder a todos los servicios proporcionados por el nodo *Master*.

49. Instalación de los paquetes necesarios Nova para la incorporación al cloud.

```
# apt-get install sysfsutils
```

```
# apt-get install nova-compute
```

50. Se modifica y configura el fichero de configuración del módulo.

```
# vim /etc/nova/nova-compute.conf
```

51. Se reinicia el servicio

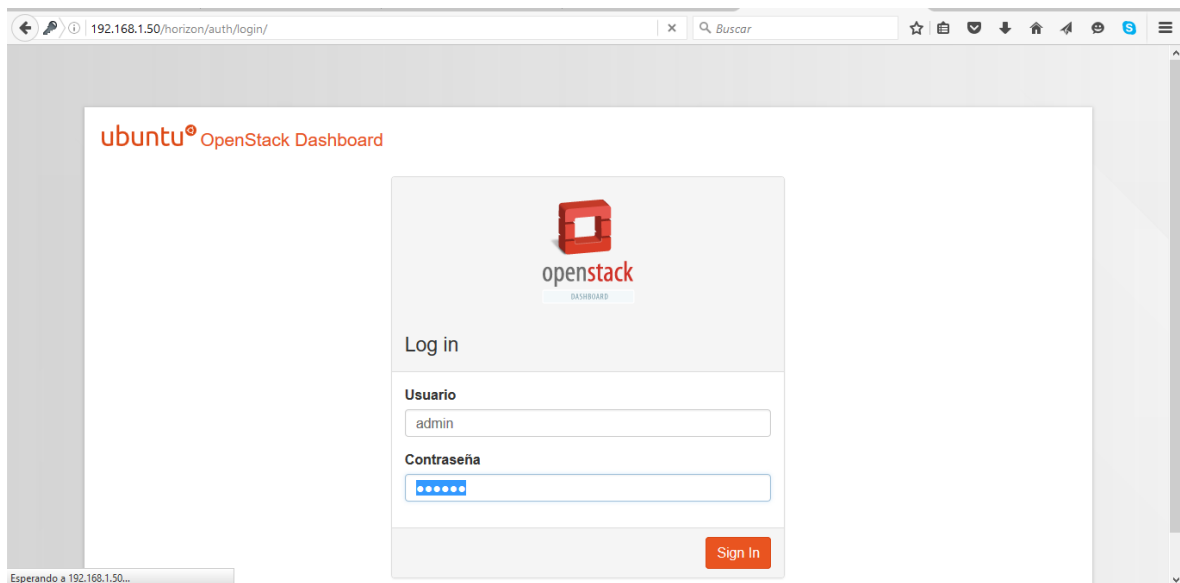
```
# service nova-compute restart
```

En este punto ya tenemos un sistema distribuido *cloud* con el *software* de soporte instalado *OpenStack*

## COMPROVACION DE TODA LA INSTALACION MEDIANTE INTERFAZ WEB

---

Una vez realizados todos estos pasos, se puede acceder desde cualquier navegador a la URL *http:192.168.1.50/horizon* y aparecerá el *Login* para entrar en *OpenStack*.



Manual 2: Login del servicio web para Openstack

## ANEXO C:

### MANUAL NUMERO 3 CLÚSTER MESSAGE PASSING INTERFACE

---

- CREACIÓN DE UN CLUSTER CON RASPBERRY, SO RASPBIAN + MPI

#### Requerimientos:

---

En este punto asumimos que tenemos el sistema operativo base **Raspbian** instalado en nuestra *SD* con el teclado configurado, la IP estática y conexión a la red. Aún así dejaré unos breves pasos.

- Descargar la última versión de la imagen del SO desde:  
<https://www.raspberrypi.org/downloads>
- Copiamos la imagen en la SD usando la opción “**Write**”, yo he usado “**Win32 Disk Imager**” <http://es.ccm.net/download/descargar-32747-win32-disk-imager>
- La insertamos en nuestra **Raspberry** y la encendemos (usr:pi pwd:raspberrypi)
- El siguiente paso será abrir el panel de configuración de nuestra máquina con el comando **raspi-config** para expandir el *File System* para recuperar el tamaño de nuestra tarjeta y configurar el teclado.
- Reiniciamos la máquina” **sudo shutdown -r now**”.
- Durante el proyecto necesitamos configurar una *IP LAN* estática para poder referenciar correctamente la máquina, además de acceder a ella remotamente, por ejemplo con **ssh** de Linux o un cliente como **Putty** para otras plataformas. Así que modificamos el fichero **/etc/network/interfaces**, normalmente la *IP LAN* que asignemos tendrá el siguiente formato:192.168.X.X. Reiniciamos el servicio **/etc/init.d/networking restart** y la máquina.

Para más detalle examinar la primera parte del **MANUAL** del Anexo A.

#### Introducción clúster MPI

---

En este manual nos centraremos en crear un clúster mediante **MPI** (“**Message Passing Interfaces**”) para ejecutar programas en paralelo con un conjunto de nodos, estos nodos serán cada una de las *Raspberrys* que poseamos.

Más sobre MPI: [http://informatica.uv.es/iiguia/ALP/materiales2005/2\\_2\\_introMPI.htm](http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm)

El esquema de clúster es el siguiente:

MÁSTER lanza los procesos a los nodos que los ejecutaran y devolverán el resultado, el Máster también puede colaborar en el proceso.

NODO cada máquina que ejecuta uno o varios procesos.

En nuestro caso el MÁSTER será un *Raspberry*, igual que los nodos, así que empezaremos por el Servidor Máster.

### Instalación del nodo MASTER

---

1. Actualizar la lista de paquetes de la memoria caché.  
**\$sudo apt-get update**
2. Descargar Fortran, un lenguaje de programación en alto nivel especializado en el cálculo numérico y la computación científica.  
**\$sudo apt-get install gfortran**
3. Creamos un directorio para almacenar los recursos utilizados.  
**\$mkdir /home/pi/mpich2**  
**\$cd ~/mpich2**
4. Descargamos la versión estable más reciente de *MPI*  
<http://www.mpich.org/downloads/>  
**\$wget <http://www.mpich.org/static/tarballs/3.2rc2/mpich-3.2rc2.tar.gz>**
5. Descomprimos el archivo.  
**\$tar xfz mpich-3.2rc2.tar.gz**
6. Crearemos una estructura de directorios para que sea más fácil la gestión de nuestro MPI.  
**\$sudo mkdir /home/rpimpi**  
**\$sudo mkdir /home/rpimpi/mpich2-install #para archivos compilados**  
**\$sudo mkdir /home/pi/mpich\_build #para archivos de herramientas.**
7. Cambiamos al directorio de las herramientas.  
**\$cd /home/pi/mpich\_build**
8. Ahora configuramos la construcción de *MPI* con el siguiente comando. Esperamos en cada acción.  
**\$sudo /home/pi/mpich2/mpich-3.2rc2/configure** -

```
prefix=/home/rpimpi/mpich2-install
```

```
$sudo make
```

```
$sudo make install
```

9. Añadimos en la variable de entorno PATH el camino de los binarios de nuestro MPI.

```
$export PATH=$PATH:/home/rpimpi/mpich2-install/bin
```

Ahora bien, nos interesa mantener el PATH al reiniciar la máquina, así que nuestra opción es editar el fichero **.profile** de nuestro **/home/pi/** para ello:

```
$vim ~/.profile
```

Añadimos lo siguiente:

```
PATH="$PATH:/home/rpimpi/mpich2-install/bin"
```

10. Comprobamos que MPI se instaló correctamente.

```
$which mpicc #Debería aparecer el path
```

```
$which mpiexec #Debería aparecer el path
```

11. Volvemos al **/home/pi** y creamos un directorio para hacer nuestras pruebas.

```
$cd ~
```

```
$mkdir mpi_testing
```

```
$cd mpi_testing
```

12. En este punto podemos probar MPI sobre un nodo mediante el comando

```
mpiexec -f <file_with_ips_master/nodes> -n <number> <app/command>
```

Creamos un fichero donde agregaremos la IP de nuestra máquina.

```
$vim machinefile
```

Añadir IP:

```
192.168.1.35
```

13. Ejecutamos

```
$mpiexec -f machinefile -n 1 hostname #el resultado tiene que ser el mismo  
que ejecutar el comando hostname
```

```
pi@raspberrypi:/home $ hostname  
raspberrypi  
pi@raspberrypi:/home $ mpiexec -f ../home/pi/mpi_testing/machinefile -n 1 hostname  
raspberrypi  
pi@raspberrypi:/home $
```

*Manual 3: Ejecución del programa machinefile.*



14. Ahora ya tenemos MPI funcionando en un único nodo, a partir de ahora este primer nodo será nuestro Master. Hacemos una última prueba ejecutando un poco de código C de ejemplo que nos ofrece el software instalado, ejecutamos.

**\$mpirun -f machinefile -n 2 ../mpich\_build/examples/cpi**

```
pi@raspberrypi:/home $ mpirun -f ../home/pi/mpi_testing/machinefile -n 1 ~/mpich_build/examples/cpi
Process 0 of 1 is on raspberrypi
pi is approximately 3.1415926544231341, Error is 0.0000000008333410
wall clock time = 0.001186
```

*Manual 3: Ejecución del programa cpi, cálculo del número pi.*

```
pi@raspberrypi:/home $ ls
pi  rpimpi
pi@raspberrypi:/home $ ls rpimpi/
mpich2-install
pi@raspberrypi:/home $ ls rpimpi/mpich2-install/
bin  include  lib  share
pi@raspberrypi:/home $ ls pi/
mpich2  mpich_build  mpi_testing
pi@raspberrypi:/home $ ls pi/mpich2/
mpich-3.2rc2  mpich-3.2rc2.tar.gz
pi@raspberrypi:/home $ ls pi/mpich_build/
config.log  config.status  doc  lib  Makefile  src
config.lt  config.system  examples  libtool  mpich-doxxygen  test
pi@raspberrypi:/home $ ls pi/mpi_testing/
machinefile
pi@raspberrypi:/home $
```

*Manual 3: Captura de la estructura actual de directorios*

## Instalación de un NODO.

---

Llegados a este punto tenemos ya una copia del MASTER (nodo principal) con todos los archivos MPI instalados correctamente, configurados y en funcionamiento, ahora debemos añadir un nodo al Clúster. Para ello, seguimos con las instrucciones. **Debemos clonar la tarjeta SD**, nuestro disco duro.

15. Apagamos nuestra Raspberry Pi Master.

**\$sudo shutdown -h now**

16. Extraemos la micro SD y la insertamos en un ordenador.
17. Copiamos la tarjeta mediante la herramienta mencionada anteriormente **“Win32 Disk Imager”**

18. Esta vez usamos la opción “**Read**” y damos nombre al fichero de backup, por ejemplo:

```
backup_node_raspbian_master.img
```

19. Quitamos la tarjeta **MASTER** del ordenador y metemos una nueva.
20. Repetimos los pasos mencionados en los **Requerimientos** desde el apartado **b.** y utilizando la imagen del backup generado en el punto anterior ( No podemos olvidar de asignar una nueva IP estática a esta máquina NODO, en mi caso le asigné 192.168.1.40).
21. Además podemos cambiar el hostname a la máquina NODO para que las pruebas sean más concluyentes. Para cambiar el hostname editamos:

```
$sudo vim /etc/hostname
```

, en mi caso

```
node1_raspberrypi
```

22. Reiniciamos la máquina:

```
$sudo shutdown -r now
```

En este punto, con las dos Raspberrys conectadas tenemos nuestro Clúster MPI, ahora configuraremos una “relación de confianza” entre ambas, para ello TODOS LOS SIGUIENTES PASOS SE EJECUTARÁN EN NUESTRO MASTER DEL CLUSTER, A NO SER QUE SE INDIQUE LO CONTRARIO.

Información sobre la seguridad en el Clúster de Raspberrys:  
<https://steve.dynedge.co.uk/2012/05/30/logging-into-a-raspberry-pi-using-publicprivate-keys/>

### **“Relaciones de confianza” CLUSTER SECURITY**

---

23. Creación de la “relación de confianza”

```
$cd ~
```

```
$ssh-keygen -t rsa -C “raspberry@raspberry”
```

Esto crea una ubicación predeterminada de `/home/pi/.ssh/id_rsa` para almacenar la clave. Nos pide introducir una frase como password, por ejemplo “**raspcluster**”, se puede dejar en blanco en ese caso no es tan seguro.

24. Este comando copia desde la máquina MASTER al nodo la clave de la relación.

```
$cat ~/.ssh/id_rsa.pub | ssh pi@192.168.1.40 "mkdir .ssh;cat >>
```

**.ssh/authorized\_keys"**

25. Para comprobar que la copia se ha hecho correctamente ejecutar estos dos comandos en el NODO.

**\$cd ~**

**\$sudo ls -la .ssh/ #deberia de aparecer el fichero **authorized\_keys**.**

26. Ahora vamos a añadir la IP del nuevo NODO al fichero machinefile del MASTER.

**\$cd mpi\_testing**

**\$vim machinefile**

Añadir IP:

**192.168.1.40**

**\$cat machinefile**

**192.168.1.35**

**192.168.1.40**

27. Ahora ya tenemos MPI funcionando con 2 nodos (**master+slave**). Podemos ejecutar en el MASTER las pruebas anteriormente realizadas y ver los cambios.

Hemos cambiado el número de procesos a lanzar a 5. Vemos que en ambos casos aparecen las referencias tanto de raspberrypi(MASTER), node1\_raspberrypi(NODE)

```
pi@raspberrypi:/home $ mpiexec -f ../home/pi/mpi_testing/machinefile -np 5 hostname
raspberrypi
raspberrypi
raspberrypi
node1_raspberrypi
node1_raspberrypi
```

*Manual 3: Nueva ejecución del programa machinefile.*

```
pi@raspberrypi:/home $ mpiexec -f ../home/pi/mpi_testing/machinefile -np 5 ~/mpich_build/examples/cpi
Process 1 of 5 is on node1_raspberrypi
Process 2 of 5 is on raspberrypi
Process 3 of 5 is on node1_raspberrypi
Process 4 of 5 is on raspberrypi
Process 0 of 5 is on raspberrypi
pi is approximately 3.1415926544231225, Error is 0.0000000008333294
wall clock time = 0.002783
pi@raspberrypi:/home $
```

*Manual 3: Nueva ejecución del programa cpi, cálculo del número pi.*

28. Ya tenemos un Clúster con dos nodos. Para añadir más nodo seguir el apartado **Instalación de CLUSTER MPI NODE** en el punto 21.

### **Probar otros ejemplos proporcionados por MPI**

---

MPI nos ofrece más ejemplos para probar el clúster, los pasos a seguir serian estos.

Desde el MASTER:

```
$pwd  
/home/pi/mpich_build/examples  
$sudo cp /home/pi/mpich2/mpich-3.2rc2/examples/hellow.c .  
$mpicc hellow.c -o hellow  
$ls  
cpi cpi.o hellow hellow.c Makefile parent parent.c  
$mpiexec -f ../home/pi/mpi_testing/machinefile -np 5  
~/mpich_build/examples/hellow
```

Desde el NODO:

```
$pwd  
/home/pi/  
$scp pi@192.168.1.35:/home/pi/mpich_build/examples/hellow ~  
$sudo mv hellow mpich_build/examples/
```

Finalmente ejecutamos el programa hellow desde el Máster.

```
$mpiexec -f ../home/pi/mpi_testing/machinefile -np 5  
~/mpich_build/examples/hellow
```

## PROGRAMAS EN C UTILIZADOS EN EL ANEXO C

### Sí paralelizado

---

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10
#define MAX_RAND 100

int matrizA[SIZE][SIZE], matrizB[SIZE][SIZE], matrizR[SIZE][SIZE];

int main (int argc, char *argv[])
{
    /* Variables */
    int numProc;
    int idProc;

    int temporal;
    int namelen;
    char hostname[MPI_MAX_PROCESSOR_NAME];
    int i, j, k;
    int from, to;

    double inicalc = 0.0, endcalc;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &idProc);
    MPI_Comm_size(MPI_COMM_WORLD, &numProc);
    MPI_Get_processor_name(hostname,&namelen);
    if (SIZE%numProc!=0) {
        if (idProc==0) printf("La matriz no es divisible por el numero de procesadores\n");
        MPI_Finalize();
        exit(-1);
    }

    from=idProc*SIZE/numProc;
    to=(idProc+1)*SIZE/numProc;

    if(idProc==0){
        int i,j;
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                matrizA[i][j] = rand()%11+2;
            }
        }
    }
}
```

```
printf("\n----- MATRIZ A -----\n");
for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
        printf("%d \t", matrizA[i][j]);
    }
    printf("\n");
}

for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
        matrizB[i][j] = rand()%11+2;
    }
}

printf("\n----- MATRIZ B -----\n");
for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
        printf("%d \t", matrizB[i][j]);
    }
    printf("\n");
}
}

MPI_Bcast(matrizB, SIZE*SIZE, MPI_INT, 0, MPI_COMM_WORLD);
if(idProc==0){
    MPI_Scatter (&matrizA[0][0], SIZE*SIZE/numProc, MPI_INT, MPI_IN_PLACE,
SIZE*SIZE/numProc, MPI_INT, 0, MPI_COMM_WORLD);
} else {
    MPI_Scatter (&matrizA[0][0], SIZE*SIZE/numProc, MPI_INT,
&matrizA[from][0], SIZE*SIZE/numProc, MPI_INT, 0, MPI_COMM_WORLD);
}

printf("Soy %s con el id %d y estoy calculando(from row %d to %d)\n", hostname,
idProc, from, to-1);

inicalc = MPI_Wtime();

for (i = from; i < to; i++) {
    for (j = 0; j < SIZE; j++) {
        temporal = 0;
        for (k = 0; k < SIZE; k++){
            temporal += matrizA[i][k] * matrizB[k][j];
            matrizR[i][j] = temporal ;
        }
    }
}

endcalc = MPI_Wtime();

if(idProc==0){
    MPI_Gather (MPI_IN_PLACE, SIZE*SIZE/numProc, MPI_INT, &matrizR[0][0],
SIZE*SIZE/numProc, MPI_INT, 0, MPI_COMM_WORLD);
}
```

```
    }else{
        MPI_Gather (&matrizR[from][0], SIZE*SIZE/numProc, MPI_INT,
&matrizR[0][0], SIZE*SIZE/numProc, MPI_INT, 0, MPI_COMM_WORLD);
    }
    if(idProc==0){
        printf("\n");
        printf("\n----- Multiplicacion A x B -----\n");
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                printf("%d \t", matrizR[i][j]);
            }
            printf("\n");
        }
        printf("\n\n");
        printf("El calculo ha tardado = %f\n", endcalc-inicalc);
    }
    MPI_Finalize();
    return 0;
}
```

## No paralelizado

---

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
```

```
int main (int argc, char *argv[])
{
```

```
    /* Variables */
    int numProc;
    int idProc;
    int F;
    int C;
    int temporal;
    int numeroAleatorio;
    int namelen;
    char hostname[MPI_MAX_PROCESSOR_NAME];
    int i, j, k;
```

```
    double initimeall = 0.0, endtimeall;
    double inicalc = 0.0, endcalc;
```

```
    MPI_Status status;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &idProc);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &numProc);
MPI_Get_processor_name(hostname,&namelen);

if(idProc==0){

    printf( "Soy %s y voy a generarte las matrices A y B\n\n", hostname );
    /*Generar matriz introducida por el usuario*/
    printf("Elige numero de filas y numero de  columnas. Se generara una matriz
aleatoria:\n");
    scanf("%d %d", &F, &C);
    printf("\nEl numero de filas es: %d\t\n", F);
    printf("El numero de columnas es: %d\t\n", C);

    initimeall = MPI_Wtime();

    int matrizA[F][C];

    for (i = 0; i < F; i++) {
        for (j = 0; j < C; j++) {
            matrizA[i][j] = rand()%11+2;
        }
    }

    printf("\n----- MATRIZ A -----\n");
    for (i = 0; i < F; i++) {
        for (j = 0; j < C; j++) {
            printf("%d \t", matrizA[i][j]);
        }
        printf("\n");
    }

    int matrizB[C][F];

    for (i = 0; i < C; i++) {
        for (j = 0; j < F; j++) {
            matrizB[i][j] = rand()%11+2;
        }
    }

    printf("\n----- MATRIZ B -----\n");
    for (i = 0; i < C; i++) {
        for (j = 0; j < F; j++) {
            printf("%d \t", matrizB[i][j]);
        }
        printf("\n");
    }

    printf("\n----- Multiplicacion A x B -----\n");

    inicalc = MPI_Wtime();
```



```
int resultado[F][F];

for (i = 0; i < F; i++) {
    for (j = 0; j < F; j++) {
        temporal = 0;
        for (k = 0; k < C; k++){
            temporal += matrizA[i][k] * matrizB[k][j];
            resultado[i][j] = temporal ;
        }
    }
}

endcalc = MPI_Wtime();
for (i = 0; i < F; i++) {
    for (j = 0; j < F; j++) {
        printf("%d \t", resultado[i][j]);
    }
    printf("\n");
}

}

if(idProc==0){
    endtimeall = MPI_Wtime();
    printf("El calculo ha tardado = %f\n", endcalc-inicalc);
    printf("El programa entero ha tardado = %f\n", endtimeall-initimeall);
}

MPI_Finalize();
return 0;

}
```